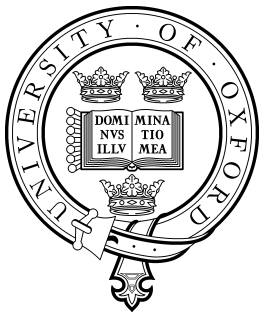
Iterative Methods for Roots of Polynomials



Wankere R. Mekwi
Exeter College
University of Oxford

A thesis submitted for the degree of
MSc in Mathematical Modelling and Scientific Computing
Trinity 2001

Abstract

We describe iterative methods for polynomial zerofinding and, specifically, the Laguerre method and how it is used in the NAG subroutine C02AFF. We also investigate a bug that has been in this subroutine for ten years. In chapter two, we give a brief survey of some zerofinding methods. These include Bairstow's method, Bernoulli's method, Graeffe's root-squaring method, Müller's method, the Newton-Raphson method and the Jenkins-Traub and Laguerre methods. In chapter three, we look at the Laguerre method as used in C02AFF in further detail, describe the behaviour of the bug and how the problem has been solved. We also describe general tests for zerofinding algorithms and results of comparisons between NAG's C02AFF and other zerofinding programs. Chapter 4 involves comparisons of C02AFF with other methods and a note on error bounds. Finally, we make our proposals and conclusions in chapter 5.

Acknowledgements

I wish to sincerely thank Prof. L. N. Trefethen for his careful supervision of this project. For reading through and correcting this work meticulously and making his wide spectrum of books and articles available to me. It has been a privilege working under him as he has been a source of inspiration to me.

Many thanks go to Dr. S. Hammarling of NAG and the entire NAG team for their cooperation which has lead to our success. Special thanks to Dr. D. J. Allwright who has been there to help at every stage in the project and for understanding the project so profoundly. I am obliged to Dr. M. Embree who was always available to help me.

My appreciations also go to the Association of Commonwealth Universities (ACU) for sponsoring me on this challenging MSc.

I want to thank my family back at home (Cameroon) for their support - dad and mum, Vicky, Kari, Nkanjoh and Kayeh you've been a source of strength during this period. I cannot name every other relative, but their support is acknowledged. Thanks to all the friends I've made on the course for bringing all the fun and, when things got tough it was good to know I wasn't alone!

Lastly and most importantly, thanks to the Lord Jesus Christ, who is 'before all things and by whom all things consist.' *Colossians 1:16*

Contents

Chapter 1

Introduction

The problem of solving the polynomial equation

$$p(x) = p_0 + p_1 x + p_2 x^2 + \ldots + p_n x^n = 0 \tag{1.1}$$

was known to the Sumerians (third millennium BC) and has greatly influenced the development of mathematics throughout the centuries. Starting with the Sumerian and Babylonian times, the study of polynomial zerofinding focused on smaller degree equations for specific coefficients [Pan97]. The solution of specific quadratic equations by the Babylonians (about 200 BC) and the Egyptians (found in the Rhind or Ahmes papyrus of the second millennium BC) corresponds to the use of the quadratic formula:

$$x_{1,2} = \frac{-p_1 \pm \sqrt{p_1^2 - 4p_0p_2}}{2p_2}. \tag{1.2}$$

A full understanding of this solution formula, however, required the introduction of negative, irrational and complex numbers. Attempts to find solution formulae which would involve only arithmetic operations and radicals succeeded in the sixteenth century for polynomials of degree 3 and 4. However, for polynomials of degree greater than 4, it was not possible[1] to find such formulae as was proved by Abel in 1827. The Galois theory was motivated by the same problem of solving (1.1) and included the proof of the nonexistence of the solution in the form of formulae. In spite of the absence of solution formulas in radicals, the *fundamental theorem of algebra* states that equation (1.1) always has a complex solution for any polynomial $p(x)$ of any positive degree n. With no hope left for the exact solution formulae, the motivation came for designing iterative algorithms for the approximate solution and, consequently, for introducing several major techniques. Actually the list of iterative algorithms proposed for approximating the solution $z_1, z_2, z_3, \ldots, z_n$ of (1.1) includes hundreds of items and encompasses about four millennia. An extensive survey of literature dealing with zerofinding is found in [McN93] and contains hundreds of references.

The Numerical Algorithms Group (NAG) specialises in developing software for the solution of complex mathematical problems. Their polynomial rootfinding subroutine C02AFF, which has had a bug for about 10 years now, is our main interest in this work. We shall describe in this work how we successfully found the bug in this subroutine and results of our comparisons with other widely used algorithms for polynomial rootfinding.

[1] In fact, Omar Khayyam, who died in 1122, a famous poet and the leading mathematician of his time, and later Leonardo de Pisa (now commonly known as Fibonacci), who died in 1250, wrongly conjectured the nonexistence of such solution formulae for $n = 3$ [Pan97].

Chapter 2

Some Zerofinding Algorithms

In this chapter, we look at some of the outstanding algorithms that have been proposed and used in the 20th century. First we define some concepts that are inherent to the subject of polynomial zerofinding.

2.1 Important Concepts

A numerical method for determining a zero of a polynomial generally takes the form of a prescription to construct one or several sequences z_n of complex numbers supposed to converge to a zero of the polynomial. As one would expect, each algorithm has its advantages and disadvantages and therefore the choice of the 'best' algorithm for a given problem is never easy.

Any reasonable algorithm must converge, i.e., the sequence generated by it should, under suitable conditions, converge to a zero of the given polynomial. An algorithm must also be designed to produce approximations to both real and complex roots of a polynomial. Other desirable properties that an algorithm may or may not have include the following:

1. *Global Convergence:* Many algorithms can be guaranteed to converge only if the starting value z_0 is sufficiently close to a zero of the polynomial. These are said to be **locally convergent**. Algorithms that do not require a sufficiently close starting value are **globally convergent**.

2. *Unconditional Convergence:* Some algorithms will only converge if the given polynomial has some special properties, e.g., all zeros simple or no equimodular zeros. These algorithms are **conditionally convergent**. If an algorithm is convergent (locally or globally) for all polynomials, it is **unconditionally convergent**.

3. *A posteriori Estimates:* In practice, any algorithm must be artificially terminated after a finite number of steps. The approximation at this stage, z_n, say, will not in general be identical to a zero ξ of the polynomial. Under such circumstances, it is desirable that we be able to calculate, from the data provided by the algorithm, a bound β_n for the error $|z_n - \xi|$ of the last approximation. A precise statement can then be made that there is at least one zero of the polynomial in the disk $|z - z_n| \leq \beta_n$. Certain algorithms incorporate the calculation of such a bound β_n in their definition, while for others it can be computed from the data given.

4. *Speed of Convergence:* The concept of *order* is frequently used as a measure of the ultimate speed of convergence of an algorithm. The order ν is defined as the supremum of all real numbers α such that

$$\limsup_{n\to\infty} \frac{|z_{n+1}-\xi|}{|z_n-\xi|^\alpha} < \infty.$$

   Newton's method, for example, has the order of convergence $\nu = 2$, which means asymptotically that the number of correct decimal places is doubled at each step. Thus, the higher the order of convergence, the faster $|z-\xi|$ converges ultimately to zero.

5. *Simultaneous Determination of All Zeros:* Most algorithms determine one zero at a time. If the zero has been determined with sufficient accuracy, the polynomial is deflated and the algorithm is applied again on the deflated polynomial. It may be desirable for practical as well as theoretical reasons [Hen74] to determine all zeros simultaneously.

6. *Cluster Insensitivity:* A major problem in the numerical determination of zeros is presented by the occasional occurrence of the 'clusters' of zeros, i.e., sets of several zeros that either coincide or are close. The performance of many otherwise excellent methods is worsened in the presence of a cluster. We therefore want methods that are insensitive to clusters.

7. *Numerical Stability:* In real life, all computing is done in the finite system of discrete and bounded numbers of a machine, instead of the field of real or complex numbers. The set of numbers provided by floating point arithmetic is finite. Thus, algorithms originally devised to work in the continuum are adapted to 'machine numbers' by the devices of rounding and scaling. Not all algorithms are equally insensitive to this adaptation. By numerical stability, we mean the lack of sensitivity to rounding and scaling operations or more precisely, the sensitivity of the algorithm should be no greater than that inherent in the ill-conditioning of the zerofinding problem.

2.2 Bairstow's Method

This method is only valid for polynomials with real coefficients. For such polynomials, we know that any complex roots occur as conjugate pairs. The method attempts to find the zeros of such polynomials by searching for pairs of zeros which generate real quadratic factors. Thus, if we define

$$P(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0, \tag{2.1}$$

where the a_i's are real, then dividing P by the real quadratic polynomial $z^2 + pz + q$, we can write

$$P(z) \equiv (z^2 + pz + q)(b_{n-2} z^{n-2} + \ldots + b_0) + Rz + S \tag{2.2}$$

where $Rz + S$ is the remainder. Equating coefficients gives

$$b_k = a_{k+2} - p b_{k+1} - q b_{k+2}, \qquad k = n-2, \ldots, 0 \tag{2.3}$$

and

$$b_{n-1} = b_n = 0.$$

Hence, the coefficients b_k can be regarded as uniquely defined functions of p and q. Also, R and S are functions of p and q and are defined (from equating coefficients) by

$$R(p,q) = a_1 - pb_0 - qb_1 \equiv b_{-1}, S(p,q) = a_0 - qb_0 \tag{2.4}$$

The solution of

$$R(p,q) = 0 \text{ and } S(p,q) = 0 \tag{2.5}$$

yields p and q such that $(z^2 + pz + q)$ is a quadratic factor of P.

Bairstow suggested that the equation (2.1) be solved by Newton's process of successive approximations [Bro75, GR67]. Let p_i and q_i, p_{i+1} and q_{i+1} denote respectively the results of the ith and $(i+1)$st steps in the iteration. A sequence (p_i, q_i) is generated by

$$\begin{bmatrix} p_{i+1} \\ q_{i+1} \end{bmatrix} = \begin{bmatrix} p_i \\ q_i \end{bmatrix} - \begin{bmatrix} \frac{\partial R}{\partial p} & \frac{\partial R}{\partial q} \\ \frac{\partial S}{\partial p} & \frac{\partial S}{\partial q} \end{bmatrix}^{-1} \begin{bmatrix} R \\ S \end{bmatrix}_{\substack{p=p_i \\ q=q_i}} \quad i = 0, 1, 2, \ldots$$

Thus

$$p_{i+1} = p_i - \frac{1}{J}\left[R\frac{\partial S}{\partial q} - S\frac{\partial R}{\partial q}\right]_{\substack{p=p_i \\ q=q_i}} \tag{2.6}$$

$$q_{i+1} = q_i - \frac{1}{J}\left[S\frac{\partial R}{\partial p} - R\frac{\partial S}{\partial p}\right]_{\substack{p=p_i \\ q=q_i}} \tag{2.7}$$

where

$$J = \begin{vmatrix} \frac{\partial R}{\partial p} & \frac{\partial S}{\partial p} \\ \frac{\partial R}{\partial q} & \frac{\partial S}{\partial q} \end{vmatrix}_{\substack{p=p_i \\ q=q_i}}$$

Differentiating the equations (2.3) and (2.4) with respect to p and q gives

$$\frac{\partial R}{\partial p} = -p\frac{\partial b_0}{\partial p} - q\frac{\partial b_1}{\partial p} - b_0,$$

$$\frac{\partial R}{\partial q} = -p\frac{\partial b_0}{\partial q} - q\frac{\partial b_1}{\partial q} - b_1$$

$$\frac{\partial S}{\partial p} = -q\frac{\partial b_0}{\partial p}$$

$$\frac{\partial S}{\partial q} = \frac{\partial b_{-2}}{\partial q} + p\frac{\partial b_{-1}}{\partial q} \tag{2.8}$$

and

$$\begin{aligned}\frac{\partial b_k}{\partial p} =& - b_{k+1} - p\frac{\partial b_{k+1}}{\partial p} - q\frac{\partial b_{k+2}}{\partial p}, \qquad k = n-3, \ldots, 0, -1 \\ \frac{\partial b_{n-2}}{\partial p} =& \frac{\partial b_{n-1}}{\partial p} = 0 \end{aligned} \tag{2.9}$$

$$\begin{aligned}\frac{\partial b_k}{\partial q} =& - b_{k+2} - p\frac{\partial b_{k+1}}{\partial q} - q\frac{\partial b_{k+2}}{\partial q}, \qquad k = n-4, \ldots, 0, -1, -2 \\ \frac{\partial b_{n-3}}{\partial q} =& \frac{\partial b_{n-2}}{\partial q} = 0 \end{aligned} \tag{2.10}$$

If we define the recurrence relation

$$\begin{aligned} d_k =& - b_{k+1} - pd_{k+1} - qd_{k+2}, \qquad k = n-3, \ldots, 0, -1 \\ d_{n-2} =& d_{n-1} = 0 \end{aligned} \tag{2.11}$$

then we have from (2.9) and (2.10) that

$$\frac{\partial b_k}{\partial p} = d_k, \quad \frac{\partial b_{k-1}}{\partial p} = d_k, \quad k = n-3, \ldots, 0, -1 \tag{2.12}$$

and

$$\begin{aligned}\frac{\partial R}{\partial p} =& d_{-1}, \quad \frac{\partial R}{\partial q} = d_0 \\ \frac{\partial S}{\partial p} =& - qd_0, \frac{\partial S}{\partial q} = d_{-1} + pd_0 \end{aligned} \tag{2.13}$$

Therefore, (2.6) and (2.7) become

$$p_{i+1} = p_i - \frac{1}{J}[b_{-1}(d_{-1} + p_i d_0) - (b_{-2} + p_i b_{-1})d_0], \tag{2.14}$$

$$q_{i+1} = q_i - \frac{1}{J}[(b_{-2} + p_i b_{-1})d_{-1} + d_0 b_{-1} q_i] \tag{2.15}$$

where

$$J = d_{-1}^2 + p_i d_0 d_{-1} + q_i d_0^2. \tag{2.16}$$

After obtaining a quadratic factor, the polynomial p is deflated and the same procedure is applied to the deflated polynomial.

This method, when it converges, is quadratically convergent. However, such convergence requires a very good initial approximation though the method uses only real arithmetic to compute even complex roots. It is limited to real polynomials and therefore is not so convenient in practice as one is usually interested in polynomials with complex coefficients as well. Unless modified, the method is quite slow to converge to quadratic factors of multiplicity greater than 1. Extensions, however, exist and can be see in [Art72].

2.3 Bernoulli's Method

Bernoulli's method exploits the connection between a linear difference equation and the zeros of its characteristic polynomial in order to find the zeros of a polynomial without knowing crude first approximations. Given the polynomial

$$p(z) = a_0 z^k + a_1 z^{k-1} + \ldots + a_k \tag{2.17}$$

with $k \geq 1$ and $a_0 a_k \neq 0$, the difference equation which has p as its characteristic polynomial is given as

$$a_0 x_n + a_1 x_{n-1} + \ldots + a_k x_{n-k} = 0. \tag{2.18}$$

Given any starting values $x_0, x_1, \ldots, x_{k-1}$, the corresponding solution of (2.18) can be found numerically by the recurrence relation:

$$x_n = -\frac{1}{a_0}(a_1 x_{n-1} + a_2 x_{n-2} + \ldots + a_k x_{n-k})$$
$$n = k, k+1, k+2, \ldots$$

If we suppose that the zeros $z_1, z_2, \ldots, z_k$ of p all have multiplicity 1 (i.e. distinct zeros) then the solution of (2.18) can be expressed in the form

$$x_n = c_1 z_1^n + c_2 z_2^n + \ldots + c_k z_k^n. \tag{2.19}$$

The c_j's can be computed if the zeros are known. But since these are not known, the c_j's are unknown. However, the quotients

$$q_n = \frac{x_{n+1}}{x_n}$$

using (2.19) are analytically represented by

$$q_n = \frac{c_1 z_1^{n+1} + c_2 z_2^{n+1} + \ldots + c_k z_k^{n+1}}{c_1 z_1^n + c_2 z_2^n + \ldots + c_k z_k^n}. \tag{2.20}$$

Furthermore, suppose p has a single dominant zero,[1] and let this be z_1 for our case (this zero will be real if the coefficients of p are real). Also, assume that the starting values of $x_0, x_1, \ldots, x_{k-1}$ of the solution $\{x_n\}$ of (2.18) are chosen so that $c_1 \neq 0$. [2] Under these assumptions, we may write (2.20) as

$$q_n = z_1 \frac{1 + \frac{c_2}{c_1}\left(\frac{z_2}{z_1}\right)^{n+1} + \ldots + \frac{c_k}{c_1}\left(\frac{z_k}{z_1}\right)^{n+1}}{1 + \frac{c_2}{c_1}\left(\frac{z_2}{z_1}\right)^{n} + \ldots + \frac{c_k}{c_1}\left(\frac{z_k}{z_1}\right)^{n}}. \tag{2.21}$$

Under our second supposition, as $n \to \infty$,

$$\left(\frac{z_j}{z_1}\right)^n \to 0 \text{ for } j = 2, 3, \ldots, k,$$

and it follows that

$$\lim_{n\to\infty} q_n = z_1. \tag{2.22}$$

Having obtained this zero, the polynomial (2.17) is deflated and the procedure repeated on the deflated polynomial. Thus, this method can only furnish one or two zeros of a given

[1] If a polynomial p of degree K has zeros $z_1, z_2, \ldots, z_k$, not necessarily distinct, then the zero z_j is called dominant if its modulus is strictly greater than the moduli of the other zeros, i.e. $|z_j| > |z_i|$ for $i \neq j$.

[2] It can be shown [Hen64] that this condition is always satisfied if the starting values are chosen so that

$$x_{-k+1} = x_{-k+2=} \ldots = x_{-1} = 0, x_0 = 1.$$

polynomial at a time, and these zeros are those of largest or smallest magnitude. So, if a zero of intermediate modulus is desired, it is necessary to compute all larger (or all smaller) zeros and then 'remove' them from the polynomial by deflation.

We have obtained the above result on the assumption that the dominant zero, z_1, is real and distinct. Nevertheless, (2.22) holds when z_1 is multiple but real. The necessary changes in the method in such a case have been described in detail in [Hen82]. Note that if there is a dominant zero that is complex (2.22) no longer holds. Also if z_2 has nearly the same magnitude as z_1 the convergence process is very slow. Hence, as a general-purpose method, this method has little in its favour. However, if the zero of largest or smallest magnitude (by considering $p(1/z)$ as described in [Hen82]) is the zero that is desired and is distinct, Bernoulli's method can be useful.

An extension of this method due to Rutishauser, with the advantage that it provides simultaneous approximations to all zeros, is the Quotient-Difference (QD) algorithm. However, unless special measures are taken, it is only linearly convergent. We do not discuss this here.

2.4 Graeffe's Root-squaring Method

Graeffe's method basically replaces the equation

$$p(z) = a_n z^n + a_{n-1} z^{n-1} + \ldots + a_1 z + a_0 \tag{2.23}$$

by an equation still of degree n, whose roots are the squares of the roots of (2.23). By iterating this procedure, the roots of unequal magnitude become more widely separated in magnitude. By separating the roots sufficiently, it is possible to calculate the roots directly from the coefficients. However, the process is not suitable for polynomials some of whose roots are of equal magnitude.

Suppose the roots of (2.23) are α_i, $i = 1, \ldots, n$ and assume $a_n = 1$. Then, writing $p_0(z)$ for $p(z)$, we have

$$p_0(z) = \prod_{i=1}^{n} (z - \alpha_i). \tag{2.24}$$

Using this we can write

$$p_1(w) = (-1)^n p_0(z) p_0(-z) = \prod_{i=1}^{n} (w - \alpha_i^2), \qquad w = z^2, \tag{2.25}$$

so that the zeros of $p_1(w)$ are precisely the squares of the zeros of $p_0(z)$. Thus the sequence

$$p_{r+1}(w) = (-1)^n p_r(z).p_r(-z), \qquad r = 0, 1, \ldots \tag{2.26}$$

is such that the zeros of each polynomial are the squares of the zeros of the previous polynomial. If the coefficients of $p_r(z)$ are denoted by $a_j^{(r)}, j = 0, 1, \ldots, n$ it can be shown that

$$a_j^{(r+1)} = (-1)^{n-j} \left[(a_j^{(r)})^2 + 2 \sum_{k=1}^{\min(n-j,j)} (-1)^k a_{j-k}^{(r)} a_{j+k}^{(r)} \right]. \tag{2.27}$$

To use the sequence of polynomials $\{p_r(z)\}$, we need a relationship between the coefficients of the polynomial and its zeros. This relationship is expressed by the equation

$$a_j^{(r)} = (-1)^{n-j} S_{n-j}(\alpha_1^{2r}, \alpha_2^{2r}, \ldots, \alpha_n^{2r}), \qquad j = 0, \ldots, n-1, \tag{2.28}$$

where $S_k(x_1, x_2, \ldots, x_n)$ is the kth symmetric function of $x-1, \ldots, x_n$. This function is defined by the equation

$$S_k(x_1, x_2, \ldots, x_n) = \sum_1^n{}_c x_{r_1} x_{r_2} \ldots x_{r_k} \tag{2.29}$$

where the notation $\sum_c$ denotes that the sum is over all combinations of k out of the digits 1 to n in the subscripts. Thus, for example,

$$a_{n-1}^{(r)} = -S_1(\alpha_1^{2r}, \alpha_2^{2r}, \ldots, \alpha_n^{2r}) = \sum_{k=1}^n \alpha_k^{2r} \tag{2.30}$$

Let

$$\alpha_k = \rho_k e^{i\phi_k} \quad k = 1, \ldots, n.$$

Suppose first that all the roots are distinct in magnitude and ordered so that

$$\rho_1 > \rho_2 > \ldots > \rho_n. \tag{2.31}$$

We write (2.30) as

$$a_{n-1}^{(r)} = -\alpha \left[1 + \sum_{k=2}^n \left(\frac{\alpha_k}{\alpha_1} \right)^{2r} \right].$$

Then using (2.31), we have

$$\lim_{r \to \infty} |a_{n-1}^{(r)}|^{1/2r} = |\alpha_1|.$$

Therefore, for sufficiently large r,

$$\rho_1 \approx |a_{n-1}^{(r)}|^{1/2r}.$$

Similarly, we have

$$a_{n-2}^{(r)} = \sum_1^n{}_c \alpha_{r1}^{2^r} \alpha_{r2}^{2^r} = \alpha_{r1}^{2^r} \alpha_{r2}^{2^r} \left[1 \; + \sum_{1,(r1,r2) \neq (1,2)}^n \left(\frac{\alpha_{r1} \alpha_{r2}}{\alpha_1 \alpha_2} \right)^{2r} \right],$$

and therefore, for sufficiently large r,

$$\rho_2 \approx \frac{1}{\rho_1} |a_{n-2}^{(r)}|^{1/2r} \approx \left| \frac{a_{n-2}^{(r)}}{a_{n-1}^{(r)}} \right|^{1/2r}.$$

Continuing in this way, we have in general

$$\rho_k \approx \left| \frac{a_{n-k}^{(r)}}{a_{n-k+1}^{(r)}} \right|^{1/2r} \qquad k = 3, \ldots, n. \tag{2.32}$$

In practice "sufficiently large r" means that we must continue the root-squaring process until the approximations to the magnitudes have stabilised to the number of decimal places we want. When the roots are all separated, then once we have the magnitudes, determining the sign is easily accomplished by inserting the magnitude into (2.23).

Graeffe's method is globally, although not unconditionally, convergent and in many cases produces all zeros simultaneously with quadratic convergence.

2.5 Müller's Method

This method extends the idea of the secant method [Gau97, Ueb95, Tra64] which works with a linear polynomial, to a quadratic polynomial.

Given three previous estimates $z^{(k-2)}$, $z^{(k-1)}$ and $z^{(k)}$, for an unknown root, a new value is computed by

$$z^{(k+1)} = z^{(k)} + h_k q_k, \tag{2.33}$$

where

$$\begin{aligned}
q_k &= \frac{-2C_k}{B_k \pm \sqrt{B_k^2 - 4A_kC_k}}, && (2.34)\\
C_k &= (1+r_k)P(z^{(k)}), && (2.35)\\
B_k &= (2r_k+1)P(z^{(k)}) - (1+r_k)^2 P(z^{(k-1)}) + r_k^2 P(z^{(k-2)}), && (2.36)\\
A_k &= r_k P(z^{(k)}) - r_k(1+r_k)P(z^{(k-1)}) + r_k^2 P(z^{(k-2)}), && (2.37)\\
h_k &= z^{(k)} - z^{(k-1)}, \text{ and} && (2.38)\\
r_k &= h_k/h_{k-1}. && (2.39)
\end{aligned}$$

The values q_k computed in (2.34) may yield too large changes for $z^{(k)}$ which possibly leads to another root and causes slow convergence [LF94]. This can be circumvented by allowing a fixed maximum increase of $|q_k|$ from one iteration step to the next. Care must also be taken when computing $P(z^{(k)})$ which is necessary to compute A_k, B_k and C_k. If an estimate of $|P(z^{(k)})|$ indicates a value greater than the maximum possible number, we choose

$$z^{(k+1)} = z^{(k)} + h_k q_k/2 \tag{2.40}$$

in place of (2.33) and repeat this until no overflow occurs. The algorithm stops whenever the actual value $|P(z^{(k)})|$ is smaller than the smallest value $|P(z_{min})|$ until now and

$$\left| \frac{z_{min} - z^{(k+1)}}{z_{min}} \right| < \epsilon \tag{2.41}$$

holds where ϵ is some small number depending on the computer accuracy. To avoid a lot of iterations where condition (2.41) fails we allow only a fixed maximum number of iterations.

Convergence for Müller's method is superlinear (1.84). However, it is one of those methods that will converge to both real and complex roots from a real initial approximation.

2.6 Newton's Method

We briefly describe a well-known iterative method for approximating the zeros of a polynomial equation. Starting with a given initial approximation x_0, a sequence $x_1, x_2, x_3, \ldots$ is computed where x_{n+1} is given by

$$x_{n+1} = x_n + h_n \tag{2.42}$$

where

$$h_n = -\frac{p(x_n)}{p'(x_n)}.$$

The iterative process can be stopped when $|h_n|$ has become less than the largest error one is willing to permit in the root.

This method is only locally convergent and will converge to complex zeros only if the initial approximation is complex. However, it can be suitably modified (see for example [RR78, §8.2]) to compute zeros of complex polynomials. When Newton's method does converge, the convergence is quadratic to roots that are simple.

Note: Experiments have shown that a good combination of Müller's and Newton's methods can produce a reliable and fast program [LF94]. Müller's method is used to compute an estimate for the polynomial. This estimate is then used as the initial approximation in Newton's method and once it converges, the polynomial is deflated and the procedure repeated. Results from such experiments have indicated that this combination is better than the eigenvalue method in speed and accuracy especially for high degree polynomials ($n > 500$).

2.7 Jenkins-Traub Algorithm

The Jenkins-Traub method (J-T) is a three-stage method that calculates the zeros of the polynomial

$$p(z) = a_0 z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n, \tag{2.43}$$

which may be written in terms of its factors as

$$p(z) = \prod_{j=1}^{k} (z - \alpha_j)^{m_j}, \tag{2.44}$$

where α_i are the zeros we wish to compute and, of course, m_i is the multiplicity of the root α_i and hence satisfies $\sum_{j=1}^{k} m_j = n$. Notice that we have assumed here that $a_0 = 1$, but this is only for convenience and no generality is lost.

Zeros are calculated one at a time and zeros of multiplicity m are found m times. The zeros are found in roughly increasing order of magnitude to avoid instability arising from deflation with a large zero [Wil63]. After each zero is found the polynomial is deflated and the algorithm applied to the deflated polynomial. In the outline of the algorithm below, we take p to be either the original polynomial or a polynomial obtained by deflation.

From (2.44), we have that

$$p'(z) = \sum_{j=1}^{k} m_j P_j(z) \tag{2.45}$$

where

$$P_j(z) = \frac{p(z)}{z - \alpha_j}, \quad j = 1, 2, \ldots, k. \tag{2.46}$$

We now generate a sequence of polynomials $H^{(\lambda)}(z)$ starting with $H^{(0)}(z) = p'(z)$, each of the form

$$H^{(\lambda)}(z) = \sum_{j=1}^{k} c_j^{(\lambda)} P_j(z) \tag{2.47}$$

with $c_j^{(0)} = m_j, j = 1, \ldots, k$. If we can choose such a sequence so that $H^{(\lambda)}(z) \to c_1^{(\lambda)} P_1(z)$, that is, so that

$$d_j^{(\lambda)} = \frac{c_j^{(\lambda)}}{c_1^{(\lambda)}} \to 0, \;\; j = 2, \ldots, k, \tag{2.48}$$

then the sequence $\{t_\lambda\}$ of approximations to α_1 can be found and t_λ defined by

$$t_{\lambda+1} = s_\lambda - \frac{p(s_\lambda)}{\widetilde{H}^{\lambda+1}(s_\lambda)} \tag{2.49}$$

where

$$\widetilde{H}^{(\lambda)}(z) = \frac{H^{(\lambda)}(z)}{\sum_{j=1}^{p} c_j^{(\lambda)}} \tag{2.50}$$

is monic and $\{s_\lambda\}$ is an arbitrary sequence of complex numbers. Indeed using (2.46) and (2.48) in (2.49), we obtain

$$\begin{aligned} t_{\lambda+1} &= s_\lambda - \frac{p(s_\lambda)\sum_{j=1}^{k} c_j^{(\lambda+1)}}{\sum_{j=1}^{k} c_j^{(\lambda+1)} P_j(s_\lambda)} \\ &= s_\lambda - \frac{P_1(s_\lambda)(s_\lambda-\alpha_1)c_1^{(\lambda+1)}\left(1+\sum_{j=2}^{k} d_j^{(\lambda+1)}\right)}{P_1(s_\lambda)c_1^{(\lambda+1)}\left(1+\sum_{j=2}^{k} d_j^{(\lambda+1)}\frac{P_j(s_\lambda)}{P_1(s_\lambda)}\right)}, \end{aligned}$$

which approaches $s_\lambda - (s_\lambda - \alpha_1) = \alpha_1$. The $H^{(\lambda)}(z)$ are generated by

$$H^{(\lambda+1)}(z) = \frac{1}{z-s_\lambda}\left[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s_\lambda)}{p(s_\lambda)}p(z)\right]. \tag{2.51}$$

Evidently, such a sequence can be generated so long as $p(s_\lambda) \neq 0$. Otherwise, s_λ is a zero of $p(z)$ and so $p(z)$ can be deflated and the process repeated. In fact, using (2.46) and (2.47) in (2.51), we find that

$$\begin{aligned} H^{(\lambda+1)}(z) &= \frac{p(z)}{z-s_\lambda}\left(\sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{z-\alpha_j} - \sum_{j=1}^{k}\frac{c_j^{(\lambda)}}{s_\lambda-\alpha_j}\right) \\ &= \sum_{j=1}^{k} c_j^{(\lambda+1)} P_j(z), \end{aligned} \tag{2.52}$$

where

$$c_j^{(\lambda+1)} = \frac{c_j^{\lambda}}{\alpha_j - s_\lambda}, \qquad j = 1, \ldots, k. \tag{2.53}$$

Hence,

$$c_j^{(\lambda+1)} = \frac{c_j^{(\lambda)}}{\alpha_j - s_\lambda} = \frac{c_j^{(\lambda+1)}}{(\alpha_j - s_\lambda)(\alpha_j - s_{\lambda-1})} = \ldots = \frac{m_j}{\prod_{t=0}^{\lambda}(\alpha_j - s_t)}, j = 1, \ldots, k \tag{2.54}$$

and if no s_t is a zero of $p(z)$, $c_j^{(\lambda)} \neq 0$ for all j. The method can be summarised as follows:

1. *Stage One:* $s_\lambda = 0$

$$\begin{aligned} H^{(0)}(z) &= p'(z), \\ H^{(\lambda+1)}(z) &= \frac{1}{z}\left[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(0)}{p(0)}p(z)\right], \lambda = 0, 1, \ldots, M-1. \end{aligned}$$

2. *Stage Two:* $s_\lambda = s$
   Take β to be a positive number such that $\beta \leq \min |\alpha_i|$ and let s_λ be such that $|s_\lambda| = \beta$ and
   $$|s_\lambda - \alpha_1| < |s - \alpha_i|, i = 2, \ldots, k.$$
   (The zero labelled α_1 depends on the choice of s_λ)
   $$H^{(\lambda+1)}(z) = \frac{1}{z-s}\left[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s)}{p(s)}p(z)\right], \lambda = M, M+1, \ldots, L-1$$

3. *Stage Three:* $s_\lambda = t_\lambda$. Take
   $$t_\lambda = s - \frac{p(z)}{\widetilde{H}^{(\lambda+1)}(z)}$$
   and let
   $$\begin{aligned} H^{(\lambda+1)}(z) &= \frac{1}{z-s_\lambda}\left[H^{(\lambda)}(z) - \frac{H^{(\lambda)}(s_\lambda)}{p(s_\lambda)}p(z)\right] \\ s_{\lambda+1} &= s_\lambda - \frac{p(s_\lambda)}{\widetilde{H}^{(\lambda+1)}(s_\lambda)}, \lambda = L, L+1, \ldots. \end{aligned}$$

Note that the termination of Stage 1 is not crucial. Indeed, stage 1 is not necessary for theoretical considerations [JT70b, RR78]. Its main purpose, however, is to accentuate the smaller zeros. Numerical experience indicates that $M = 5$ is suitable.

The parameter s in stage 2 (fixed shift process) is chosen so that $|s| = \beta, \beta = \min |\alpha_j|, j = 1, \ldots, k$ and so that $|s - \alpha_1| < |s - \alpha_j|$, $j = 2, \ldots, k$. β is the unique positive zero of the polynomial
$$z^n + |a_1|z^{n-1} + \ldots + |a_{n-1}|z - |a_n|,$$
and is easily computed by Newton-Raphson iteration.

The second stage is intended to separate equimodular or at least almost equimodular zeros. In practice, L is determined when
$$|t_\lambda - t_{\lambda-1}| \leq \frac{1}{2}|t_{\lambda-1}| \text{ and } |t_{\lambda-1} - t_{\lambda-2}| \leq \frac{1}{2}|t_{\lambda-2}|$$
Stage 3 is terminated when the computed value of the polynomial at s_λ is less than or equal to some bound on the round off error in evaluating $p(s_\lambda)$. This stage has been shown to be equivalent to Newton-Raphson iteration applied to a sequence of rational functions converging to a linear polynomial [JT70b].

The J-T algorithm has the desirable feature that it is restriction-free, i.e., it converges (globally) for any distribution of zeros. This has been rigorously proved in [JT70b]. The algorithm is fast for all distributions of zeros. Also, few critical decisions have to be made by the program which implements the algorithm. Shifting is incorporated into the algorithm itself in a natural and stable way. Shifting breaks equimodularity and speeds convergence.

2.8 Eigenvalues of Companion Matrix

This method is a very accurate method for computing zeros of a polynomial. Let
$$p(z) = a_0 + a_1 z + \ldots + a_{n-1}z^{n-1} + a_n z^n, \tag{2.55}$$

and suppose (without loss of generality) $a_n = 1$. The companion matrix associated with this polynomial is the $n \times n$ matrix [TT94, EM95, Tre01]

$$C = \begin{pmatrix} 0 & & & & -a_0 \\ 1 & 0 & & & -a_1 \\ & 1 & 0 & & -a_2 \\ & & \ddots & & \vdots \\ & & & 1 & -a_{n-1} \end{pmatrix}, \tag{2.56}$$

and has the characteristic polynomial [EM95]

$$P_C(z) \equiv \det(zI - C) = p(z).$$

Thus, finding the zeros of (2.55) is equivalent to computing the eigenvalues of C. If λ is a root of (2.55), then λ is an eigenvalue of C and the associated left eigenvector v is given by

$$v = (1, \lambda, \lambda^2, \ldots, \lambda^{n-1}). \tag{2.57}$$

The eigenvalues of C can be found using the QR algorithm after C is first 'balanced' by a diagonal similarity transformation in a standard fashion due to Parlett and Reinsch. Experiments [LF94] have shown this method to be very accurate when compared with other methods. However, the fact that this method uses $O(n^2)$ storage and $O(n^3)$ time as compared to $O(n)$ storage and $O(n^2)$ time for methods designed specifically for computing zeros of polynomials [Mol91], becomes quite important for higher degree polynomials [LF94]. This is the algorithm used by MATLAB's `roots`.

2.9 Laguerre's Method

Let $p(z)$ be a polynomial with roots $r_1, r_2, \ldots, r_n$ and let z approximate the root r_j for some fixed j. This method uses $p(z)$, $p'(z)$ and $p''(z)$ to obtain a better approximation for the root r_j. Define derivatives of $\log p(z)$ as follows:

$$S_1(z) = \frac{p'(z)}{p(z)} = \sum_{i=1}^{n} \frac{1}{z - r_i}, \tag{2.58}$$

$$S_2(z) = -S_1'(z) = \frac{(p'(z))^2 - p(z)p''(z)}{(p(z))^2} = \sum_{i=1}^{n} \frac{1}{(z - r_i)^2}. \tag{2.59}$$

We write

$$\alpha(z) = \frac{1}{z - r_j}, \beta(z) + \delta_i(z) = \frac{1}{z - r_i} \quad \text{for } i = 1, 2, \ldots, n, i \neq j, \tag{2.60}$$

where β is the mean of the collection $\frac{1}{z-r_i}$, $i \neq j$. Clearly, then,

$$\sum_{i=1, i \neq j}^{n} \delta_i = 0.$$

Define

$$\delta^2 = \sum_{i=1, i\neq j}^{n} \delta_i^2. \tag{2.61}$$

Using (2.60) and (2.61), we may write (2.58) and (2.59) in the form

$$S_1 = \alpha + (n-1)\beta \tag{2.62}$$

and

$$S_2 = \alpha^2 + (n-1)\beta^2 + \delta^2. \tag{2.63}$$

Eliminating β from (2.62) and (2.63) and solving for α gives

$$\alpha = \frac{S_1 \pm \sqrt{(n-1)[nS_2 - S_1^2 - n\delta^2]}}{n}. \tag{2.64}$$

Substituting α from (2.60) in (2.64) yields

$$r_j = z - \frac{n}{S_1 \pm \sqrt{(n-1)[nS_2 - S_1^2 - n\delta^2]}}, \tag{2.65}$$

where S_1, S_2 and δ^2 are evaluated at z. Setting $\delta^2 = 0$, yields the Laguerre iteration formula

$$z_j^{(k+1)} = z_j^{(k)} - \frac{n}{S_1 \pm \sqrt{(n-1)[nS_2 - S_1^2]}}, \tag{2.66}$$

where S_1 and S_2 are evaluated at $z_j^{(k)}$, the current approximation to r_j and are defined by (2.58) and (2.59) respectively.

We note here that one Laguerre step requires more calculation than one step of any of the previously described methods, since we must compute the first and second derivatives of p at $z_j^{(k)}$. However, when there are a priori approximations to the zeros available, the reduction in the number of iterations with Laguerre more than compensates for the extra calculation. It is known (see for example [HP77, HPR77, RR78]) that if the zero is simple, then the method is cubically convergent locally. For multiple roots, the convergence is linear.

Certain modifications of this method have been shown to improve convergence [HPR77]. One modification that is fourth order convergent to simple roots is the iterate

$$z_j^{(k+1)} = z_j^{(k)} - \frac{n}{S_1 \pm \sqrt{(n-1)(nS_2 - S_1^2 - n\delta_j^2)}},$$

where

$$\delta_j^2 = \sum_{i=1, i\neq j}^{n} \left[\frac{1}{z_j^{(k)} - z_i^{(k)}} - \beta_j \right]^2$$

and

$$\beta_j = \frac{1}{n-1} \sum_{i=1, i\neq j}^{n} \frac{1}{z_j^{(k)} - z_i^{(k)}}.$$

Though Laguerre's method is exclusively for polynomial zerofinding, it belongs to a class of methods for more general that includes other methods such as Newton's and Ostrowski's [HP77]. In the next chapter, we shall explain how a modification of this method is used in the NAG subroutine C02AFF.

2.10 Algorithms used by some Software

Below is a table that summarises the algorithms used by some of the most popular numerical software programs.

Source	**Name of Program**	**Method**
NAG F77 library	C02AFF	Modified Laguerre
MATLAB	`roots`	Eigenvalues of balanced companion matrix
HSL	PA16	Madsen-Reid
IMSL	CPOLY	Jenkins-Traub
Mathematica	`NSolve`	Jenkins-Traub
Numerical Recipes	`zroots`	Laguerre

Table 2.1: Algorithms used by some commonly used numerical software.

Chapter 3

Correction of a bug in the NAG subroutine C02AFF

The Numerical Algorithms Group (NAG), established since 1970, specialises in developing software for the solution of complex mathematical problems. NAG's products span many computing languages and platforms and their subroutines are classified broadly to cover areas such as time series analysis (G13), non parametric statistics (G08), random number generators (G05), eigenvalues and eigenvectors (F02), function minimisation and maximisation (E04), summation of series (C06), zeros of polynomials (C02) etc. The heart of the company is the FORTRAN 77 library, soon to be released in Mark 20, which has evolved over the course of thirty years.

Central to this project is the C02AFF subroutine for finding zeros of a univariate polynomial[1]. This subroutine was observed to work badly for certain polynomials and/or classes of polynomials as we shall soon describe. To make things clear, we shall describe precisely how the Laguerre method described in section 2.9 is used in C02AFF.

3.1 C02AFF - A modified Laguerre method

C02AFF is NAG's subroutine for computing the zeros of a complex polynomial, i.e., a polynomial with complex coefficients. Its sister subroutine C02AGF computes roots of real polynomials. Both subroutines use a variant of Laguerre's method which we shall describe shortly. C02AFF is itself a 'dummy' routine which calls another subroutine, C02AFZ, in which the modified Laguerre method is implemented. (We shall refer more to C02AFZ in due course.) The modified method is based on the work of Brian Smith [Smi67]. For a description of the modified Laguerre method, we consider the polynomial

$$\begin{aligned} p(z) &= a_0 z^n + a_1 z^{n-1} + \ldots + a_{n-1} z + a_n \qquad (3.1) \\ &= \sum_{i=0}^{n} a_i z^{n-i} \\ &\equiv a_0 \prod_{j=1}^{n} (z - z_j), \qquad (3.2) \end{aligned}$$

[1] C02AFF was first included in the NAG libraries in 1990 and replaced the former C02ADF which served the same purpose, but was based on a method of Grant and Hitchins.

where the a_i's are complex and, in addition, we require that $a_0 a_n \neq 0$ so that the polynomial is of degree n. The cases $n \leq 2$ and polynomials whose leading or trailing coefficients vanish are treated separately. No iteration is needed for $n = 2$ or $n = 1$ since these can be obtained directly from well-known closed formulas. Thus, we assume $n \geq 3$ in what follows.

The coefficients of the polynomial are first scaled upward so that underflow does not occur when the polynomial is evaluated near zero. C02AFZ attempts to start the iteration process at the origin. If this is not an acceptable iterate (we shall see shortly what the necessary conditions are), the subroutine attempts to find a suitable iterate in an annulus about the origin known to contain the smallest zero of p. The inner radius of this annulus, R_1, is the Cauchy lower bound and is the smallest positive zero of

$$S(z) = \sum_{j=0}^{n-1} |a_j| z^{n-j} - |a_n|. \tag{3.3}$$

It is computed by a Newton-Raphson iteration.

The radius of the outer circle, R_2, is the minimum of the geometric mean of the magnitudes of the zeros G, the Fejer bound F,the Cauchy upper bound and the Laguerre bound W, where

$$G = \left| \frac{a_n}{a_0} \right|^{1/n}$$

and $W = \sqrt{n}|\mathcal{L}(\xi)|$. The Fejer bound, F, at the point ξ is the magnitude of the zero z_s of smaller magnitude of the quadratic equation

$$\frac{p''(\xi)}{2n(n-1)}\mu^2 + \frac{p'(\xi)}{n}\mu + \frac{p(\xi)}{2} = 0. \tag{3.4}$$

Thus, $F = |z_s|$.

The Laguerre step $\mathcal{L}(\xi)$ is related to the zero z_s by

$$\mathcal{L}(\xi) = \frac{z_s}{z_s \frac{(n-2)p'(\xi)}{np(\xi)} + n - 1}. \tag{3.5}$$

The origin is accepted as an initial iterate if the Laguerre step from the origin lies within the outer circle of the annulus. A trial point z_0 in this annular region is accepted as an initial iterate if the next iterate $z_1 = z_0 + \mathcal{L}(z_0)$ lies within the annulus.

Once an initial iterate has been found, subsequent iterates are determined by the following conditions. For $j = 0, 1, \ldots$

1. $z_{j+1} = z_j + L(z_j)$, **and** $|p(z_j)| > |p(z_{j+1})|$ where $L(z_j)$ may be a modified Laguerre step, and

2. $z_{j+1} + \mathcal{L}(z_{j+1})$ lies inside a circular region about the iterate z_j of radius F known to contain a zero of $p(z)$, i.e., $\mathcal{L}(z_{j+1})| \leq F$.

The modified Laguerre step is

$$L(z_{j+1}) = \begin{cases} \mathcal{L}(z_{j+1}) & \text{if } |\mathcal{L}(z_{j+1})| \leq F/2, \\ \\ \frac{F\mathcal{L}(z_{j+1})}{2|\mathcal{L}(z_{j+1})|} & \text{if } F/2 < |\mathcal{L}(z_{j+1})| \leq F. \end{cases} \tag{3.6}$$

This step may be further modified if the first criterion is not satisfied. If $|p(z_j) + L(z_j)| \geq |p(z_j)|$ then $L(z_j)$ is halved and (1) is re-tested. This is repeated until both conditions are satisfied.

The iteration procedure stops when the polynomial value at an iterate becomes smaller than a bound on the rounding error in the polynomial value evaluated at that iterate (this is computed within the subroutine C02AFY).

3.2 The Bug

Peter Brazier-Smith, working for Topexpress Ltd in 1991, noticed a bug in C02AFF when computing the dispersion relation for vibrations of an infinite fluid-loaded cylindrical shell. The shell was modelled with Arnold and Warbuton's 8th order thin shell model, and this resulted in an 8th order polynomial equation to be solved. This was reported to NAG by David Allwright[2], who was the NAG site representative for Topexpress at the time. Investigating the bug further, Allwright came up with the case

$$z^3 + iz^2 + a, \tag{3.7}$$

where a is any complex number taken from the region shown in figure 3.1. The bug also appears for polynomials of the form $z^3 - iz^2 - b$, where b is from the reflection along the x-axis of the region in figure 3.1. Indeed, suppose we wish to compute the roots of the

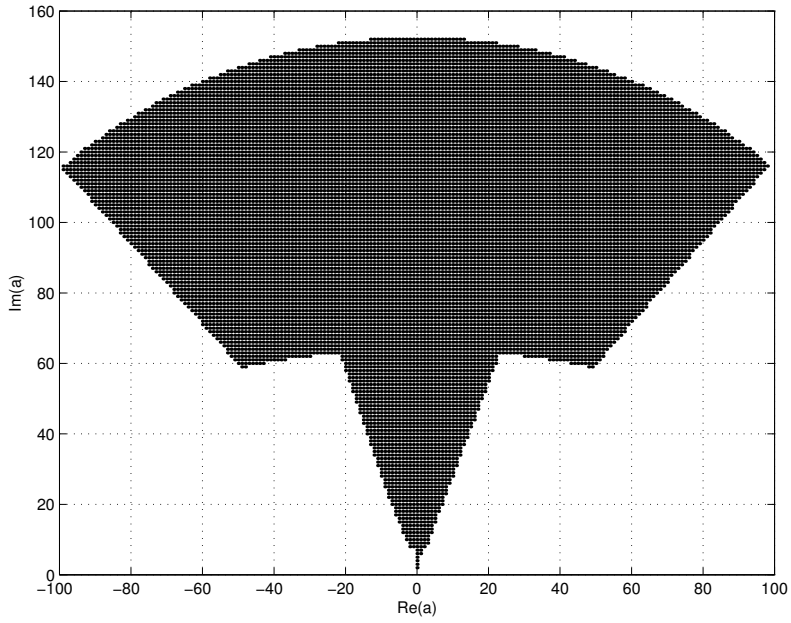


Figure 3.1: Region for trailing coefficient a for which C02AFF fails

polynomial

$$p(z) = z^3 + iz^2 + 20i. \tag{3.8}$$

[2]The bug in C02AGF was found by David Allwright, when working on a problem in optimal control of surge and stall in rotating compressors, also at Topexpress. He also came up with the example $(z^2 + az + b)^3$ for which C02AGF fails.

MATLAB's `roots` tells us that the three zeros of p are

```
 -2.31235262825422 - 1.70935513678553i
  2.31235262825422 - 1.70935513678553i
                 0 + 2.41871027357107i
```

An attempt to solve this using C02AFF returns

```
 ** The method has failed. This error is very unlikely to occur.
 ** Please contact NAG.
 ** ABNORMAL EXIT from NAG Library routine C02AFF: IFAIL =     2
 ** NAG hard failure - execution terminated
 Note: IEEE floating-point exception flags raised:
    Inexact;  Underflow;
 See the Numerical Computation Guide, ieee_flags(3M)
```

It was also noticed that for monic polynomials of degree 3, with the other coefficients being random, the subroutine failed at an average rate of about 4×10^{-4} (Allwright, personal communications). Some polynomials took a long time to compute the roots of certain of these polynomials. In recent experiments, the longest took 555.75 seconds ≈ 9.25 minutes for a polynomial of degree just 3! Apart from these, C02AFF is an apparently perfect routine, as our comparison with MATLAB's `roots` has shown.

Our strategy for tracking down the source of this bad behaviour in the NAG code has been to simplify the code as much as possible whilst preserving the properties of the bug. This has enabled us to understand C02AFF in much greater detail as well as to home in on the bug. We also mimicked the behaviour of this code by writing an equivalent piece of MATLAB code, MC02AFZ. Our MATLAB code failed to fail. Thus, we were able to tell, eventually, when and where failure set into C02AFF.

We showed that our MATLAB equivalent of C02AFZ worked correctly by ensuring that it also computed the roots of 'good' polynomials correctly. For example, computing the roots of $z^3 - 5z^2 + iz - 1$ we have for MC02AFZ

```
>> mc02afz([1 -5 1i -1])
ans =
  5.04672583446957 - 0.19483090323035i
 -0.03676011243829 + 0.55236250815270i
 -0.00996572203128 - 0.35753160492235i
```

C02AFF returns

```
  5.0467258344696   -0.19483090323035
 -3.6760112438289D-02    0.55236250815270
 -9.9657220312807D-03   -0.35753160492235
```

and MATLAB's `roots` returns

```
>>roots([1 -5 1i -1])
ans =
  5.04672583446957 - 0.19483090323035i
 -0.03676011243829 + 0.55236250815270i
 -0.00996572203128 - 0.35753160492235i.
```

Indeed both C02AFF and MC02AFZ use just two iterations to obtain the first root of this polynomial. In this and other examples where MC02AFZ and C02AFF were used, we observed agreement to many digits of the iterates along the way to convergence.

Further investigations into C02AFZ led us into C02AFY — the subroutine responsible for evaluating a polynomial, its first derivative and half its second derivative at a complex point. It also returns an estimate on the error in the polynomial value at this point. We found out that it evaluated half of the second derivative correctly, but failed to return this correct value because both the real and imaginary parts of this result were erroneously assigned the real part! Needless to say how catastrophic this typographical error can be.

Bug in C02AFY

The line
`PDPRIM(2) = WR`
should be replaced by
`PDPRIM(2) = WI`

Hence, if the polynomial evaluated at a point was purely imaginary, the subroutine returned `(0.,0.)`. This became a problem as this value eventually became the leading coefficient of the polynomial we described in equation (3.4) for computing the Fejer bound. The outcome was that one of the roots of this quadratic equation was assigned some large value. This led, of course, to repeated halving in an attempt to satisfy the first condition of the modified Laguerre. This is why a follow up of the failure case reveals over a thousand unsuccessful iterations!

We reported the discovery of this bug to NAG on 15 August 2001 and it was immediately corrected in time for the Mark 20 release of the library.

3.3 Testing Zerofinding Programs

According to [JT74] which is a standard guide to testing zerofinding programs, it is essential in testing these that one checks for: program robustness, convergence difficulties, specific weaknesses of the algorithm and reliability and efficiency. We summarise the requirements of each of these below.

3.3.1 Checking program robustness

Program robustness here refers to the ability of a program to degrade gracefully near the boundary of the problem space where the algorithm applies. To achieve this, it is essential to ask whether the program:

- tests if a leading coefficient is zero or 'nearly' zero. If it is exactly zero, the degree of the polynomial is incorrect;
- tests if a trailing coefficient is zero, or 'nearly' zero. If it is exactly zero, there is a zero of the polynomial at the origin. It can be detected directly and the problem reduced to one of lower degree;
- handles lower degree cases properly;
- scales coefficients to avoid exponent underflow and overflow difficulties;

- specifies a finite computation; and
- provides a failure exit or a testable failure flag.

3.3.2 Testing for Convergence Difficulties

In section 2.1, we discussed the importance of convergence of an algorithm. The issue here is: how do we decide when to terminate the sequence and accept the value obtained? And, if the sequence is not converging, or is converging slowly, how do we detect this and what do we do? A satisfactory way of ending a converging sequence of iterates is to stop when the polynomial value becomes dominated by the roundoff error in evaluating the polynomial. A number of polynomials have been suggested [JT74] to check issues dealing with convergence. The polynomial

$$P_1(z) = B(z^3 - z^2 - A^2z + A^2) = B(z - A)(z + A)(z - 1) \tag{3.9}$$

can be used with large and small A to test that large and small zeros do not cause the termination criterion to fail and with B large and small to ensure that large and small polynomial coefficients do not similarly cause failures.

$$P_2(z) = \prod_{i=1}^{r}(z - i) \tag{3.10}$$

with zeros at $1, 2, \ldots, r$ where r is chosen small enough that coefficients of the polynomial are exactly representable in the precision used, can be used to test convergence for ill-conditioned polynomials.

$$P_3(z) = \prod_{i=1}^{r}(z - 10^{-i}) \tag{3.11}$$

with zeros at $10^{-1}, 10^{-2}, \ldots, 10^{-r}$ where r is chosen small enough that the constant term does not overflow, may be used to test whether a termination criterion based on roundoff error analysis fails.

Multiple zeros or nearly multiple zeros cause convergence difficulties for many algorithms. The following polynomials have been suggested to check the performance of the program on multiple or nearly multiple zeros.

$$\begin{aligned}
P_4(z) &= (z - .1)^3(z - .5)(z - .6)(z - .7), && (3.12)\\
P_5(z) &= (z - .1)^4(z - .2)^3(z - .3)^2(z - .4), && (3.13)\\
P_6(z) &= (z - .1)(z - 1.001)(z - .998)(z - 1.00002)(z - .99999), && (3.14)\\
P_7(z) &= (z - .001)(z - .01)(z - .1)(z - .1 + Ai)(z - .1 - Ai)(z - 1)(z - 10), && (3.15)
\end{aligned}$$

with A chosen to be $0, 10^{-10}, 10^{-9}, 10^{-8}, 10^{-7}, 10^{-6}$, and

$$P_8(z) = (z + 1)^5. \tag{3.16}$$

Together with P_8, the polynomial

$$P_9(z) = (z^{10} - 10^{-20})(z^{10} + 10^{20}) \tag{3.17}$$

which has two sets of equimodular zeros, may be used to test an algorithm's behaviour towards equimodular zeros.

3.3.3 Defects in Program Implementing a Particular Algorithm

We make mention here of the deflation process. Deflation using the Horner recurrence [Wil63] is unstable if a zero considerably larger in modulus than a smaller zero is removed first. The polynomials

$$P_{10}(z) = (z - A)(z - 1)(z - A^{-1}), \tag{3.18}$$

with $A = 10^3, 10^6$, or 10^9, and

$$P_{11}(z) = \prod_{k=1-M}^{M} (z - e^{\frac{ik\pi}{2M}}) \prod_{k=M}^{3M} (z - .9e^{\frac{ik\pi}{2M}}) \tag{3.19}$$

with $M = 15, 20, 25$, can be used to check stability of deflation.

3.3.4 Assessment of Performance

By doing tests with randomly generated polynomials, one can gather statistics on the performance of the program with respect to the reliability, accuracy and cost prediction. There are many ways that one may use to generate random polynomials and particular choices may affect the quality of the performance evaluation greatly. We shall not pursue this further here (see [JT74]), but we make mention of the kind of random polynomials used to test how widely applicable a program may be, since the zeros vary widely. These polynomials are monic and have their other coefficients chosen randomly by taking the mantissa and exponents from separate uniform distributions, i.e., coefficients of the form

$$a_1 \times 10^{e_1} + ia_2 \times 10^{e_2}, \tag{3.20}$$

where a_i and e_i $(i = 1, 2)$ are drawn from the uniform distribution on the intervals $[-1, 1]$ and $[-10, 10]$ respectively (see [TT94]).

3.4 Tests on C02AFF

The bug found in C02AFY seems to have solved all the problems that have been previously encountered while using C02AFF. Here we describe tests carried out to vindicate this statement.

3.4.1 Reported Failures

NAG provided us with some of the failures that were reported over the last decade. The modified version of C02AFF passed this test with no problem at all.

3.4.2 Tests with Random Polynomials

We already indicated in 3.2 that a failure rate of about 4×10^{-4} was observed from experiments with degree 3 monic polynomials and with other coefficients random and complex. Mention was also made of the extremely long time taken to compute roots of one of some polynomials. Experiments conducted with the corrected C02AFF indicate the absence of these obnoxious behaviours. Indeed, the whole process of computing the roots of 10^6 polynomials took ≈ 5.5 minutes.

Chapter 4

Comparisons of C02AFF with other Methods

Here we describe experiments carried out to compare NAG's C02AFF (with the bug corrected) with other commonly used zerofinders. We did our comparisons by producing scatter plots of the errors in a given pair of zerofinders. For these plots, we used a random sample of one hundred degree-10 monic polynomials with coefficients like those described in (3.20).

4.1 Method

Our codes for PA16 [1] and CPOLY [2] were taken from the Web. Our first zerofinder, CPOLY, uses the Jenkins-Traub algorithm (section 2.7). The second zerofinder we compare with is MATLAB's ROOTS, which constructs the associated companion matrix of the polynomial. PA16, our third zerofinder, uses the method of Madsen-Reid to compute zeros. Finally, `zroots`, taken from *Numerical Recipes* [PFTV86], uses Laguerre's method as described in section 2.9.

First we compute the "exact" roots of each polynomial by computing the eigenvalues of the associated companion matrix in quadruple precision ($\approx$ 34 digits) using a LAPACK subroutine for computing the eigenvalues of a general complex matrix. The rest of our computations were carried out in double precision ($\approx$ 16 digits). First we make sure that our code computes roots of polynomials correctly. The roots of the polynomial

$$p(z) = (z - 1/3)(z + 1/3)(z - i)(z + 1/4)(z - 2),$$

are given as

```
(-.19259299443872358530559779425849 27D-33,0.10000000000000000000000000000000 00D+01)
(-.33333333333333333333333333333351D+00,-.18084867394108778958529990235885 19D-32)
(-.24999999999999999999999999999988D+00,0.15335497160429613704556511576975 98D-32)
(0.33333333333333333333333333333332D+00,0.40041616071866135115848286901461 43D-33)
(0.20000000000000000000000000000000 00D+01,-.21674502155573590641294172190279 08D-33),
```

by our code, and these are close enough to the exact roots. We used MATLAB to compute the errors for the various zerofinders and create their scatter plots.

[1] See *http* : *//hsl.rl.ac.uk/acuk/hslforacuk.html/*.

[2] Taken from TOMS algorithm 419 in *http* : *//www.netlib.org/*.

4.2 Results

Figure 4.1 shows the scatter plot for the error in CPOLY against error in C02AFF for a hundred degree-10 polynomials with random coefficients in the sense spelled out in section 3.3.4. It indicates that both methods show reasonably close accuracy in their approximations to the zeros of the random polynomials. However, our experiments show that CPOLY takes less time to compute zeros.

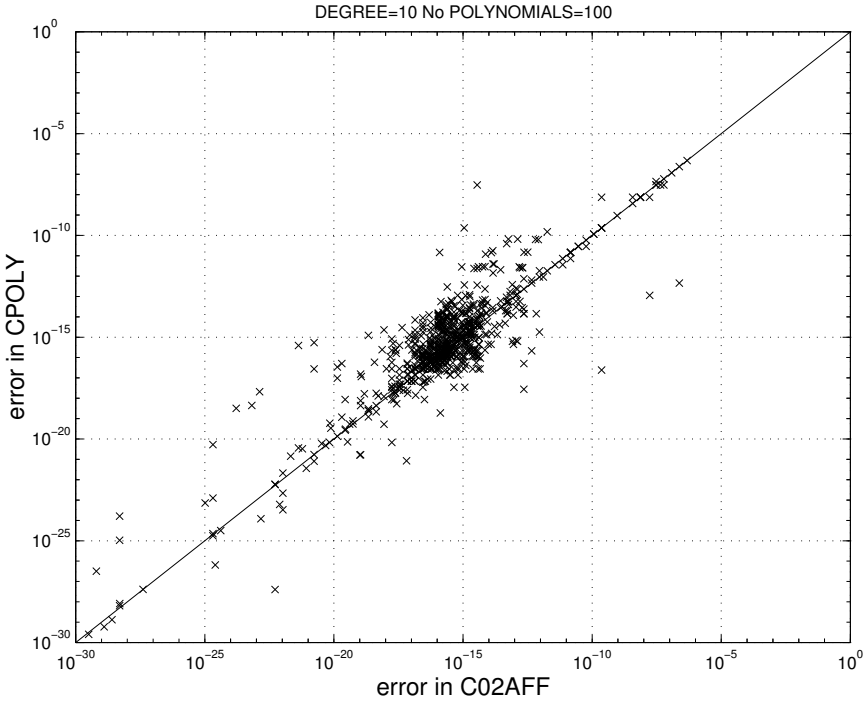


Figure 4.1: Scatter plot showing error in CPOLY against error in C02AFF.

A comparison of C02AFF and `roots` is shown in figure 4.2. The plot suggests that C02AFF is more accurate than `roots` in most cases. The scatter plot suggests that `roots` can only get 10^{-16} accuracy relative to the largest zeros, whereas C02AFF can get 10^{-16} accuracy relative to the zero in question.

Our comparison of C02AFF with Numerical Recipe's `zroots` which uses Laguerre's method, reveals that C02AFF is a better zerofinder. HSL's PA16 turns out to be more accurate than C02AFF in many cases. The scatter plot for this is shown in figure 4.3.

Figure 4.4 shows a summary of the time taken for each of the programs to compute the roots of a hundred polynomials of varying degree. It is clear from this that `roots` takes the longest time in all cases. PA16 takes the least time to compute the roots in practically all cases we tried.

Note: Our attempt to compare computation times of the various programs for polynomials of even higher degree led us to, apparently, another bug in C02AFF. From experiments with random polynomials of degree 40 and higher, we noticed that C02AFF was unable to compute the zeros for some of the 100 polynomials. A follow up of this phenomenom led us into the subroutine C02AGX, but time has not allowed us to further investigate this behaviour.

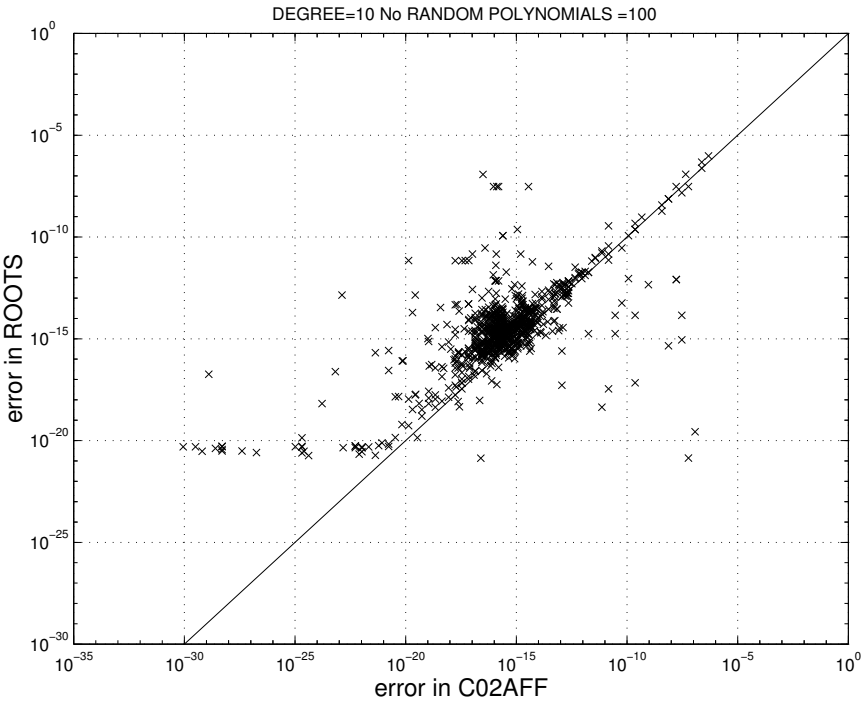


Figure 4.2: Scatter plot showing error in ROOTS against error in C02AFF.

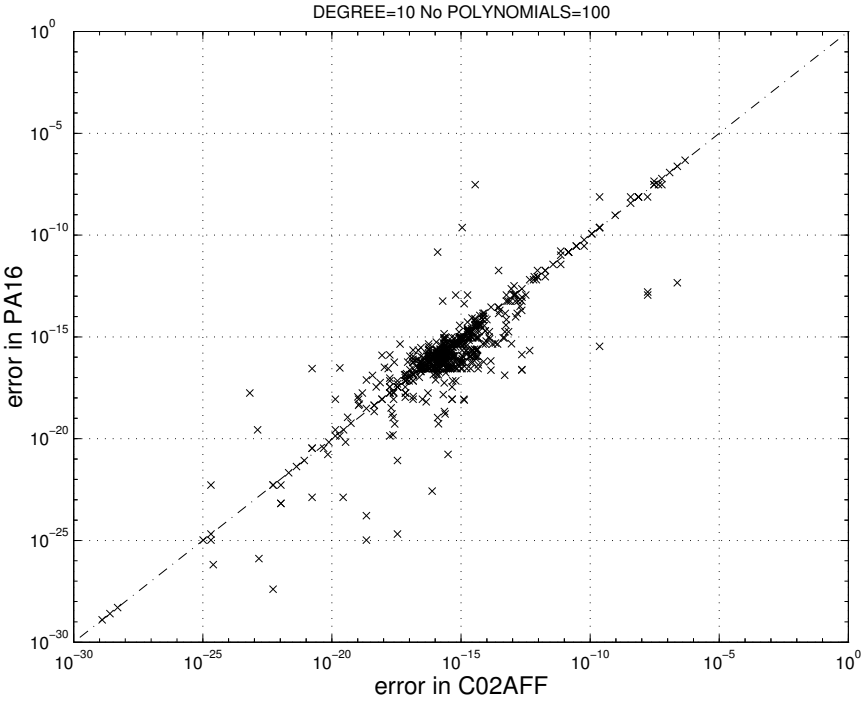


Figure 4.3: Scatter plot showing error in PA16 against error in C02AFF.

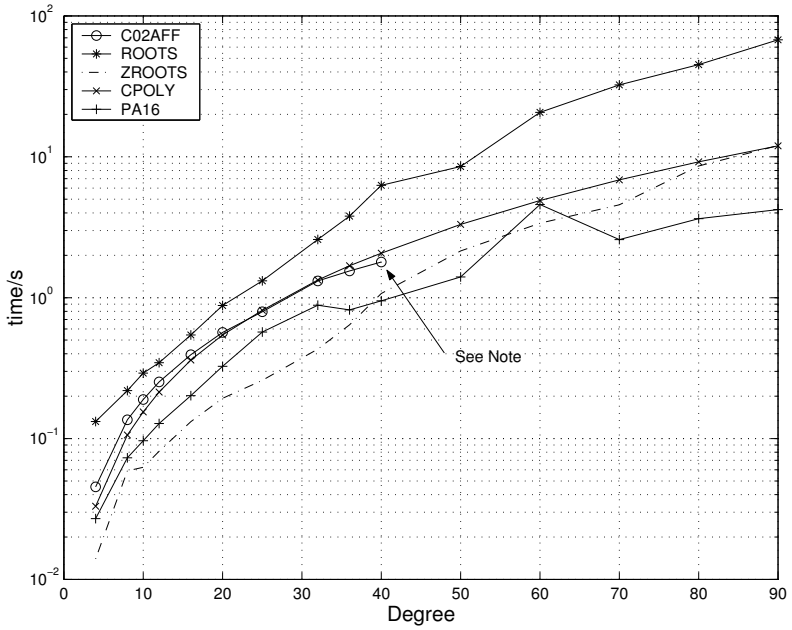


Figure 4.4: Computation time for 100 polynomials

From these results (see appendix for more experimental results), we would strongly recommend NAG's C02AFF for any zerofinding problems. With the bug corrected, it is both efficient and reliable. However, it must be admitted that PA16 appears to be even better on both counts.

4.3 A Note on Error Bounds

We are interested in finding bounds on how close the exact roots lie to out numerical approximations.

Suppose z^* is an approximate root and write

$$f(z^* + \zeta) = g(\zeta) = b_0\zeta^n + b_1\zeta^{n-1} + \ldots + b_n, \tag{4.1}$$

so that $b_n = f(z^*)$ and each $b_{n-k} = f^{(k)}(z^*)/k!$. If we are approximating an isolated root, b_n is small but b_{n-1} is not. But if we are approximating a p-fold repeated root, or a cluster of p close roots, then $b_n, b_{n-1}, \ldots,$ are small and b_{n-p} is the first non-small coefficient. In this case, we use Rouché's theorem [Cop62].

Theorem 4.3.1 (Rouché) *If $f(z)$ and $g(z)$ are two functions regular within and on a closed contour C, on which $f(z)$ does not vanish and also $|g(z)| < |f(z)|$, then $f(z)$ and $f(z) + g(z)$ have the same number of zeros within C.*

Rouché's theorem tells us, basically, that "small perturbations to $f(z)$ have the same number of zeros".

Let us write

$$\begin{aligned} g(\zeta) &= b_{n-p}\zeta^p + \sum_{0\le j\le n, j\neq p} b_{n-j}\zeta^j \\ &\equiv F(\zeta) + G(\zeta). \end{aligned} \tag{4.2}$$

In the disc $|\zeta| \le R$, $F(\zeta)$ has a zero of multiplicity p at $\zeta = 0$. Rouché's theorem tells us that if $|G(\zeta)| \le |F(\zeta)|$ on the circle $|\zeta| = R$, then $F(\zeta) + G(\zeta) = g(\zeta) = f(z^* + \zeta)$ also has p zeros in the disc $|z - z^*| < R$.

Now,

$$\begin{aligned} |F(\zeta)| &= |b_{n-p}|R^p \qquad \text{and} \\ |G(\zeta)| &= \Big| \sum_{0\le j\le n, j\neq p} b_{n-j}\zeta^j \Big| \\ &\le \sum_{0\le j\le n, j\neq p} |b_{n-j}|R^j \end{aligned}$$

If $|b_{n-p}|R^p > \sum_{0\le j\le n, j\neq p} |b_{n-j}|R^j$ then the second requirement of the theorem is met. In order for this to hold for $p = 1$, we must find R such that

$$|b_{n-1}|R > |b_n| + |b_{n-2}|R^2 + \ldots + |b_0|R^n.$$

If $|b_n|$ is small enough, the aim is to find the smallest $R > 0$ satisfying this and this will give a bound on how close the actual root is to z^*. However, if $|b_{n-1}|$ is too small, there may be no R that satisfies this. We proceed to $p = 2$ and attempt to find R such that

$$|b_{n-2}|R^2 > |b_n| + |b_{n-1}|R + |b_{n-3}|R^3 + \ldots + |b_0|R^n,$$

is satisfied. If this can be solved, the least $R > 0$ will give a bound on a disc around z^* in which we know 2 roots.

If this does not hold either, we continue in a similar manner for $p = 3, \ldots$. At some point, we hope to find a p for which it works and then (choosing R as small as possible) we have a bound on the p-cluster.

Chapter 5

Conclusion

Our main aim in this project was to find a bug which has been in NAG's zerofinding program, C02AFF, for about a decade. This we have successfully done. We have done tests on the corrected program that show that it is a good program. Apart from these, we have performed comparisons with other zerofinders and our experiments have revealed that C02AFF is one of the most efficient and reliable programs available. However, we also make note of an apparent bug which affects the computation of roots of higher degree polynomials by C02AFF. A little follow-up suggests that this may be directly related to the subroutine C02AGX, but we have not been able to further investigate this behaviour.

We have also made a note on error bounds for computed solutions. Apart from looking more closely at the method we propose and other methods, further work could be done on the effective implementation of these bounds and incorporating this into the C02AFF code.

Appendix A

Equivalent MATLAB codes for C02AFZ and C02AFY

Below are the equivalent MATLAB codes for C02AFZ and C02AFY. These codes are, however, simplified, omitting some of the error-checking features of the originals.

A.1 mc02afz.m

```
% A MATLAB version of the NAG subroutine C02AFZ
% for computing the zeros of polynomials using
% Laguerre's method.
%
function [z] = mc02afz(f);
ndeg = length(f) - 1; %degree of polynomial
DU = f;
n = ndeg; N = n+1;
ihalf = 0; ispir=0; iter = 0;
cauchy = 0; %Region containing smallest
%zero has not been computed
while n>2
   small = 1e-3; bigone=1.0001;
   smlone=0.99999; rconst=1.445;
   onepqt=1.25; gama=1; theta=2;
   if ~cauchy
      rtn = sqrt(n);
      G = exp((log(abs(DU(N)))-log(abs(DU(1))))/n+small);
      cr = DU(N-1)/DU(N);
      ctemp = 2*DU(N-2)/(n*(n-1));
      cf2 = DU(N-1)*2/n;
      tmp = roots([ctemp cf2 DU(N)]);
      c = tmp(2); cf1 = tmp(1);
      cr = cr*(n-2)/n;
      ctemp = cr*c + n-1;
      cdiro = c/ctemp;
      abdiro = abs(cdiro);
      G = min(G,bigone*min(abs(c),rtn*abdiro));  G = G(1);
      R = G;
      S = bigone*G; %upper bound for magnitude
%of smallest zero
      deflat(1:N) = abs(DU(1:N));
      while R < S
         T = real(deflat(1));
```

```
      S = 0;
      for i=1:n-1
         S = R*S + T;
         T = R*T + real(deflat(i+1));
      end
      S = R*S + T;
      T = (R*T - real(deflat(N)))/S;
      S = R;
      R = R - T;
   end
   cauchy = 1;
   upperb = min(rconst*n*R, G);
   lowerb = smlone*S;
end
fejer = upperb;
G = upperb;
cdir = cdiro; abdir = abdiro; ratio = abdir/G;
dzn = 0;
fn = abs(DU(N));
f0 = fn;
spiral = 0; startd=0; contin= 1;
while contin
   iter = iter + 1;
   if ratio > theta
      if startd
         ihalf = ihalf + 1;
         abscl = abscl*0.5; cl = cl*0.5;
         dx = abs(real(dzn))+abs(imag(dzn));
         if (dx+abscl ~= dx)
            dzn = dz0 + cl;
         else
            if fn >= err*n^2
               sprintf('Contact Wankere \n **Unlikely Failure')
            end
            contin = 0;
         end
      else
         ispir = ispir + 1;
         if spiral
            c = cspir*dzn;
         else
            spiral = 1;
            cspir = -onepqt/n + 1i;
            abscl = lowerb/n^2;
            ctemp = cdir/abdir;
            c = ctemp*lowerb;
         end
         dzn = c;
      end
   else
      startd = 1;
      if (ratio > gama) & (startd | spiral | lowerb <= G)
         ratio = gama/ratio;
         cdir = cdir*ratio;
         abdir = abdir*ratio;
      end
      G = fejer; cl = cdir;
      abscl = abdir;
      f0 = fn; dz0 = dzn;
```

```
         dzn = dz0 + cl;
      end
      [deflat,cf,cf1,cf2,err] = mc02afy(DU,dzn);
      fn = abs(cf);
      if cf == 0  % A root has been found
         contin = 0;  % Exit iteration loop
      else
         if (fn >= f0) & startd
            ratio = theta*bigone;
         else
            cr = cf1/cf;
            cf2 = cf2*2/((n-1)*n);
            ctemp = cf1*2/n;
            tmp = roots([cf2 ctemp cf]);
            c = tmp(2); cf1 = tmp(1);
            fejer = abs(c);
            cr = cr*(n-2)/n;
            ctemp = c*cr + n-1;
            cdir = c/ctemp;
            abdir = abs(cdir);
            ratio = abdir/G;
            fejer = min(rtn*abdir, fejer);
            dx = abs(real(dzn))+abs(imag(dzn));
            if (abdir <= 1e-15) contin = 0; end
         end
      end
   end
   DU = deflat;
   z(n) = dzn;
   N = N-1; n = n-1;
   cauchy = 0;
end
if n==2
   tmp = roots([DU(1) DU(2) DU(3)]);
   z(1) = tmp(1);
   z(2) = tmp(2);
else
   if n==1
      z(1) = -DU(2)/DU(1);
   else
      R = Inf;
   end
end
z = z(:);
```

A.2 mc02afy.m

```
function [defl,pz,p1z,p2z,error] = mc02afy(p,dx);
%Evaluate a polynomial,  its first derivative
%and half of its second derivatives at a point.
%Compute error in polynomial value at point.
%
n = length(p) - 1;
deps = 1.11e-16;
absx = abs(dx);
defl = zeros(1,length(p));
dv = p(1);
w = 0;
```

```
defl(1) = p(1);
defl(2) = p(2) + dx*defl(1);
for i=3:n+1
   w = dv + dx*w;
   dv = defl(i-1) + dx*dv;
   defl(i) = p(i) + dx*defl(i-1);
end
error = (2/3)*abs(p(1));
for i = 2:n+1
   error = abs(defl(i)) + absx*error;
end
error = 16*deps*abs(defl(n+1))+3*absx*error;
pz = polyval(p, dx);
df = (n:-1:1).*p(1:end-1);
d2f = (n-1:-1:1).*df(1:end-1);
p1z = polyval(df, dx);
p2z = 0.5*polyval(d2f,dx);
```

Appendix B

Codes for testing C02AFF

These are some of the codes we used to test C02AFF.

B.1 Reported bugs.

This is code was obtained from NAG and contains programs that were reported for faulty behaviour for both C02AGF and C02AFF.

```
*     reported_bugs.f
      PROGRAM TEST
C     .. External Subroutines ..
      EXTERNAL         EX1, EX10, EX11, EX12, EX13, EX14, EX15, EX16,
     *                 EX17, EX18, EX19, EX2, EX3, EX4, EX5, EX6, EX7,
     *                 EX8, EX9
C     .. Parameters ..
      INTEGER          NIN, NOUT
      PARAMETER        (NIN=5,NOUT=6)
C     .. Local Scalars ..
      INTEGER          PROBNO
C     .. Executable Statements ..
   20 WRITE (NOUT,FMT=*)
     *  'Choose a problem betwen 1 and 19. Type 0 to stop.'
      READ (NIN,FMT=*) PROBNO
      IF (PROBNO.EQ.1) THEN
         CALL EX1
      ELSE IF (PROBNO.EQ.2) THEN
         CALL EX2
      ELSE IF (PROBNO.EQ.3) THEN
         CALL EX3
      ELSE IF (PROBNO.EQ.4) THEN
         CALL EX4
      ELSE IF (PROBNO.EQ.5) THEN
         CALL EX5
      ELSE IF (PROBNO.EQ.6) THEN
         CALL EX6
      ELSE IF (PROBNO.EQ.7) THEN
         CALL EX7
      ELSE IF (PROBNO.EQ.8) THEN
         CALL EX8
      ELSE IF (PROBNO.EQ.9) THEN
         CALL EX9
      ELSE IF (PROBNO.EQ.10) THEN
         CALL EX10
```

```
      ELSE IF (PROBNO.EQ.11) THEN
         CALL EX11
      ELSE IF (PROBNO.EQ.12) THEN
         CALL EX12
      ELSE IF (PROBNO.EQ.13) THEN
         CALL EX13
      ELSE IF (PROBNO.EQ.14) THEN
         CALL EX14
      ELSE IF (PROBNO.EQ.15) THEN
         CALL EX15
      ELSE IF (PROBNO.EQ.16) THEN
         CALL EX16
      ELSE IF (PROBNO.EQ.17) THEN
         CALL EX17
      ELSE IF (PROBNO.EQ.18) THEN
         CALL EX18
      ELSE IF (PROBNO.EQ.19) THEN
         CALL EX19
      ELSE
         STOP
      END IF
      GO TO 20
*
      END
*
      SUBROUTINE EX1
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,9), TEMP(8), W(36), ZR(2,8)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ==============================
C     DAVID ALLWRIGHT - PROBLEM NO. 1
C     ==============================
      DATA             A/1.D0, 0.D0, 3.048D0, -0.7038D0, 12.31D0,
     *                 -5.661D0, 24.26D0, -23.93D0, 44.54D0, -76.69D0,
     *                 25.68D0, -180.3D0, 13.20D0, -348.3D0, -96.07D0,
     *                 -398.5D0, -206.0D0, -531.8D0/
C     .. Executable Statements ..
      N = 8
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, 8
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,8)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
C
```

```
      SUBROUTINE EX2
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,9), TEMP(8), W(36), ZR(2,8)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ==============================
C     DAVID ALLWRIGHT - PROBLEM NO. 2
C     ==============================
      DATA             A/1.D0, 0.D0, -0.422D0, 0.158D0, 0.335D0,
     *                 -0.109D0, -0.117D0, 0.0576D0, 6.9711D0,
     *                 -0.03083D0, -8.0434D0, 1.105057D0, 0.475D0,
     *                 -2.004112D0, -0.46545D0, 0.180345D0, -0.024512D0,
     *                 -0.12094D0/
C     .. Executable Statements ..
      N = 8
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, 8
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,8)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX3
*
*     Program to illustrate a bug in C02AGF reported by D. Allwright,
*     Cambridge Computer Laboratory, 16-October-1991.
*
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(0:3), W(8), Z(2,3)
C     .. External Subroutines ..
      EXTERNAL         C02AGF
C     .. Executable Statements ..
*     CALL A00AAF
*
      A(0) = 0.10000000000000000000D+01
      A(1) = 0.62479757587255957407D+00
      A(2) = 0.13012400360540585242D+00
      A(3) = 0.90334624461698726644D-02
      N = 3
      IFAIL = -1
*
      CALL C02AGF(A,N,.TRUE.,Z,W,IFAIL)
*
      WRITE (6,FMT=*) 'C02AGF: SCALE = .TRUE. : IFAIL = ', IFAIL
```

```
      WRITE (6,FMT=99999) (Z(1,I),Z(2,I),I=1,N)
      IFAIL = -1
*
      CALL C02AGF(A,N,.FALSE.,Z,W,IFAIL)
*
      WRITE (6,FMT=*) 'C02AGF: SCALE = .FALSE. : IFAIL = ', IFAIL
      WRITE (6,FMT=99999) (Z(1,I),Z(2,I),I=1,N)
*
99999 FORMAT (2D16.8)
      END
*
      SUBROUTINE EX4
*
*     Test program supplied by David Allwright 2-July-1992.
*
C     .. Local Scalars ..
      DOUBLE PRECISION A, B
      INTEGER          IA, IA0, IA1, IB, IB0, IB1, IFAIL, IJK, NC
C     .. Local Arrays ..
      DOUBLE PRECISION ACO(0:6), WORK(14), Z(2,6)
C     .. External Subroutines ..
      EXTERNAL         C02AGF
C     .. Intrinsic Functions ..
      INTRINSIC        DBLE, REAL
C     .. Executable Statements ..
      IA0 = 1
      IA1 = 256
      IB0 = -256
      IB1 = 256
*     WRITE (*,*) 'IA0,IA1,IB0,IB1 ?'
*     READ (*,*) IA0, IA1, IB0, IB1
      NC = (IA1-IA0+1)*(IB1-IB0+1)
      IJK = 0
      DO 40 IA = IA0, IA1
         A = DBLE(IA)
         DO 20 IB = IB0, IB1
            B = DBLE(IB)
            ACO(0) = 1.D0
            ACO(1) = 3.D0*A
            ACO(2) = 3.D0*(A**2+B)
            ACO(3) = A*(A**2+6.D0*B)
            ACO(4) = B*ACO(2)
            ACO(5) = ACO(1)*B**2
            ACO(6) = B**3
            IFAIL = 1
*
            CALL C02AGF(ACO,6,.TRUE.,Z,WORK,IFAIL)
*
            IF (IFAIL.EQ.0) IJK = IJK + 1
   20    CONTINUE
   40 CONTINUE
      WRITE (*,FMT=99999) ' C02AGF solves ', IJK, ' problems out of ',
     *  NC, ' ( = ', 100.0*REAL(IJK)/REAL(NC), ' %)'
*
99999 FORMAT (A,I8,A,I8,A,F5.1,A)
      END
*
      SUBROUTINE EX5
C     .. Local Scalars ..
```

```
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,4), TEMP(3), W(16), ZR(2,4)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ================================
C     DAVID ALLWRIGHT - PROBLEM NO. 5
C     ================================
      DATA             A/1.0D0, 0.0D0, -0.2265061220743257D+01,
     *                 -0.8876934120839897D+00, 0.1640518549907148D+01,
     *                 0.1657501114089819D+01, -0.2499153712796033D+00,
     *                 -0.7456147554629142D+00/
C     .. Executable Statements ..
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX6
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,23), TEMP(23), W(92), ZR(2,22)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ================================
C     ANDY LAM - PROBLEM NO. 1
C     ================================
      DATA             A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA             A(1,2), A(2,2)/6.692848309097203D-02,
     *                 -1.066815130745114D-02/
      DATA             A(1,3), A(2,3)/-7.086518965761863D-02,
     *                 2.230960516166786D-02/
      DATA             A(1,4), A(2,4)/6.303294768752449D-02,
     *                 -3.031411710233499D-02/
      DATA             A(1,5), A(2,5)/-5.148428562317170D-02,
     *                 3.552615504729231D-02/
      DATA             A(1,6), A(2,6)/3.423188895476926D-02,
     *                 -3.495326989239094D-02/
      DATA             A(1,7), A(2,7)/-2.282316337587525D-02,
     *                 3.089446038231147D-02/
```

```
      DATA              A(1,8), A(2,8)/1.043284095744426D-02,
     *                  -2.526122727713401D-02/
      DATA              A(1,9), A(2,9)/-3.207698906589943D-03,
     *                  1.618302686284827D-02/
      DATA              A(1,10), A(2,10)/-5.410782528762875D-03,
     *                  -4.184080580933460D-03/
      DATA              A(1,11), A(2,11)/-1.082608426110527D-03,
     *                  -6.433079945966239D-03/
      DATA              A(1,12), A(2,12)/2.140987191276102D-03,
     *                  1.807538373240856D-02/
      DATA              A(1,13), A(2,13)/-1.211010060428513D-02,
     *                  -2.698206652012696D-02/
      DATA              A(1,14), A(2,14)/1.446411522717839D-02,
     *                  3.518316442419946D-02/
      DATA              A(1,15), A(2,15)/-2.790265151409987D-02,
     *                  -3.939162263019953D-02/
      DATA              A(1,16), A(2,16)/3.769131566749238D-02,
     *                  4.025718875500140D-02/
      DATA              A(1,17), A(2,17)/-5.213611832843504D-02,
     *                  -3.657153266401287D-02/
      DATA              A(1,18), A(2,18)/6.318369421581575D-02,
     *                  3.077611824152049D-02/
      DATA              A(1,19), A(2,19)/-7.735627110673225D-02,
     *                  -2.059571828298433D-02/
      DATA              A(1,20), A(2,20)/7.945490302665662D-02,
     *                  9.238282766534648D-03/
      DATA              A(1,21), A(2,21)/-7.798948625689881D-02,
     *                  2.692352710187064D-03/
      DATA              A(1,22), A(2,22)/6.538705609654054D-02,
     *                  -1.231778873447496D-02/
      DATA              A(1,23), A(2,23)/-5.832573500768336D-02,
     *                  2.118306322814731D-02/
C     .. Executable Statements ..
*     CALL A00AAF
      N = 22
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL, (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (' IFAIL=',I4,8(/I4,3D16.7))
      END
*
      SUBROUTINE EX7
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,23), TEMP(23), W(92), ZR(2,22)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ================================
C     ANDY LAM - PROBLEM NO. 2
C     ================================
```

```
      DATA              A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA              A(1,2), A(2,2)/6.709008051116012D-02,
     *                  -1.105667830526196D-02/
      DATA              A(1,3), A(2,3)/-6.711247568226415D-02,
     *                  2.327404711680193D-02/
      DATA              A(1,4), A(2,4)/6.086150080218387D-02,
     *                  -3.166060949647874D-02/
      DATA              A(1,5), A(2,5)/-4.746169484011257D-02,
     *                  3.643474248477278D-02/
      DATA              A(1,6), A(2,6)/3.239700122706570D-02,
     *                  -3.667555858748441D-02/
      DATA              A(1,7), A(2,7)/-2.309093492605298D-02,
     *                  3.243568747240444D-02/
      DATA              A(1,8), A(2,8)/8.225779627381511D-03,
     *                  -2.665012539426458D-02/
      DATA              A(1,9), A(2,9)/-5.955953037094856D-03,
     *                  1.733181018550921D-02/
      DATA              A(1,10), A(2,10)/-5.051006621539410D-03,
     *                  -4.946482861095421D-03/
      DATA              A(1,11), A(2,11)/-4.994821509480170D-03,
     *                  -4.918252056948132D-03/
      DATA              A(1,12), A(2,12)/-2.661039956662787D-05,
     *                  1.856654584803523D-02/
      DATA              A(1,13), A(2,13)/-1.605587605278142D-02,
     *                  -2.721643640191550D-02/
      DATA              A(1,14), A(2,14)/1.431026334885683D-02,
     *                  3.588511356628187D-02/
      DATA              A(1,15), A(2,15)/-3.083505326912570D-02,
     *                  -3.984473771114714D-02/
      DATA              A(1,16), A(2,16)/3.717926367237125D-02,
     *                  4.194406205981662D-02/
      DATA              A(1,17), A(2,17)/-5.397677423606877D-02,
     *                  -3.711836065175791D-02/
      DATA              A(1,18), A(2,18)/6.318584679761217D-02,
     *                  3.222471177787703D-02/
      DATA              A(1,19), A(2,19)/-7.489069732708209D-02,
     *                  -2.163460421216798D-02/
      DATA              A(1,20), A(2,20)/7.931128414218512D-02,
     *                  9.766520016847981D-03/
      DATA              A(1,21), A(2,21)/-7.628032879105192D-02,
     *                  1.939202993846361D-03/
      DATA              A(1,22), A(2,22)/6.557098208202312D-02,
     *                  -1.327846589866429D-02/
      DATA              A(1,23), A(2,23)/-5.717600020339203D-02,
     *                  2.208546380737488D-02/
C     .. Executable Statements ..
*     CALL A00AAF
      N = 22
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL, (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (' IFAIL=',I4,8(/I4,3D16.7))
      END
*
      SUBROUTINE EX8
```

```
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,23), TEMP(23), W(92), ZR(2,22)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ================================
C     ANDY LAM - PROBLEM NO. 3
C     ================================
      DATA             A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA             A(1,2), A(2,2)/7.299717662709504D-02,
     *                 -1.251603085718758D-02/
      DATA             A(1,3), A(2,3)/-6.692829479084827D-02,
     *                 2.554480198192204D-02/
      DATA             A(1,4), A(2,4)/5.813406400365923D-02,
     *                 -3.435329359739291D-02/
      DATA             A(1,5), A(2,5)/-4.661340524475002D-02,
     *                 3.897400500670847D-02/
      DATA             A(1,6), A(2,6)/3.189537465131881D-02,
     *                 -3.971463209682351D-02/
      DATA             A(1,7), A(2,7)/-2.347222022450406D-02,
     *                 3.541977449538532D-02/
      DATA             A(1,8), A(2,8)/6.386012638991536D-03,
     *                 -2.817337272458770D-02/
      DATA             A(1,9), A(2,9)/-9.247442901437751D-03,
     *                 1.829678170485129D-02/
      DATA             A(1,10), A(2,10)/-3.181724350590678D-03,
     *                 -5.210472267566396D-03/
      DATA             A(1,11), A(2,11)/-4.042882525153831D-03,
     *                 -5.545338123709728D-03/
      DATA             A(1,12), A(2,12)/-5.079281944515824D-03,
     *                 2.269768910343822D-02/
      DATA             A(1,13), A(2,13)/-1.689026749663972D-02,
     *                 -3.271918287492030D-02/
      DATA             A(1,14), A(2,14)/1.873875002862358D-02,
     *                 4.090417193682402D-02/
      DATA             A(1,15), A(2,15)/-3.621946498825187D-02,
     *                 -4.446562197952801D-02/
      DATA             A(1,16), A(2,16)/3.807037929920166D-02,
     *                 4.726608485426496D-02/
      DATA             A(1,17), A(2,17)/-5.711044814259666D-02,
     *                 -4.258180109493363D-02/
      DATA             A(1,18), A(2,18)/6.493499815683595D-02,
     *                 3.685503603526334D-02/
      DATA             A(1,19), A(2,19)/-7.687055250545507D-02,
     *                 -2.492035912034419D-02/
      DATA             A(1,20), A(2,20)/7.995516520303647D-02,
     *                 1.310209904630223D-02/
      DATA             A(1,21), A(2,21)/-7.856518559290735D-02,
     *                 7.930764827738515D-04/
      DATA             A(1,22), A(2,22)/7.124482367933885D-02,
     *                 -1.456490551167847D-02/
      DATA             A(1,23), A(2,23)/-6.234466635505629D-02,
     *                 2.467891415118185D-02/
C     .. Executable Statements ..
```

```
*     CALL A00AAF
      N = 22
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL, (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (' IFAIL=',I4,8(/I4,3D16.7))
      END
*
      SUBROUTINE EX9
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,23), TEMP(23), W(92), ZR(2,22)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ================================
C     ANDY LAM - PROBLEM NO. 4
C     ================================
      DATA             A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA             A(1,2), A(2,2)/8.826547101919578D-02,
     *                 -1.414081968118106D-02/
      DATA             A(1,3), A(2,3)/-7.626459818385396D-02,
     *                 2.842270278133602D-02/
      DATA             A(1,4), A(2,4)/6.007807422271309D-02,
     *                 -3.632306063903809D-02/
      DATA             A(1,5), A(2,5)/-4.931801747112107D-02,
     *                 4.124866238616326D-02/
      DATA             A(1,6), A(2,6)/3.460912384784597D-02,
     *                 -4.376586918299159D-02/
      DATA             A(1,7), A(2,7)/-2.774769547328051D-02,
     *                 3.982435937329466D-02/
      DATA             A(1,8), A(2,8)/9.623668578675439D-03,
     *                 -3.303223147036237D-02/
      DATA             A(1,9), A(2,9)/-1.492626590194510D-02,
     *                 2.330373745030098D-02/
      DATA             A(1,10), A(2,10)/1.008979443592338D-03,
     *                 -9.117106753497844D-03/
      DATA             A(1,11), A(2,11)/-7.390370723224149D-03,
     *                 -2.310499485959538D-03/
      DATA             A(1,12), A(2,12)/-2.880503997552897D-03,
     *                 2.152375196652239D-02/
      DATA             A(1,13), A(2,13)/-2.123062733513988D-02,
     *                 -3.340448218406404D-02/
      DATA             A(1,14), A(2,14)/2.385860078578356D-02,
     *                 4.199245958636058D-02/
      DATA             A(1,15), A(2,15)/-4.132064378044115D-02,
     *                 -4.730288951651626D-02/
      DATA             A(1,16), A(2,16)/4.040909544333124D-02,
     *                 5.103048140306666D-02/
      DATA             A(1,17), A(2,17)/-6.018355316422513D-02,
     *                 -4.702958063167249D-02/
```

```
      DATA             A(1,18), A(2,18)/6.626630190380043D-02,
     *                 4.170523573778244D-02/
      DATA             A(1,19), A(2,19)/-7.856296020873610D-02,
     *                 -2.915845194741102D-02/
      DATA             A(1,20), A(2,20)/8.344257422934534D-02,
     *                 1.794841118976563D-02/
      DATA             A(1,21), A(2,21)/-9.211228910190803D-02,
     *                 -2.317303532165138D-03/
      DATA             A(1,22), A(2,22)/9.023511232887800D-02,
     *                 -1.466465308674939D-02/
      DATA             A(1,23), A(2,23)/-8.241393803805482D-02,
     *                 2.732938335090737D-02/
C     .. Executable Statements ..
      N = 22
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX10
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,15), TEMP(14), W(60), ZR(2,14)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ==============================
C     RICHARD MARTIN - PROBLEM NO. 1
C     ==============================
      DATA             A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA             A(1,2), A(2,2)/-0.218929D0, -0.111694D0/
      DATA             A(1,3), A(2,3)/-0.041137D0, 0.053219D0/
      DATA             A(1,4), A(2,4)/0.039693D0, -0.133413D0/
      DATA             A(1,5), A(2,5)/0.023298D0, 0.010632D0/
      DATA             A(1,6), A(2,6)/0.027063D0, -0.076606D0/
      DATA             A(1,7), A(2,7)/0.100735D0, -0.021368D0/
      DATA             A(1,8), A(2,8)/0.071725D0, -0.109003D0/
      DATA             A(1,9), A(2,9)/-0.152468D0, -0.017280D0/
      DATA             A(1,10), A(2,10)/0.044497D0, 0.110741D0/
      DATA             A(1,11), A(2,11)/-0.050943D0, 0.189754D0/
      DATA             A(1,12), A(2,12)/-0.038635D0, 0.111387D0/
      DATA             A(1,13), A(2,13)/0.017052D0, 0.062966D0/
      DATA             A(1,14), A(2,14)/0.078129D0, -0.026399D0/
      DATA             A(1,15), A(2,15)/-0.039685D0, -0.085970D0/
C     .. Executable Statements ..
*     CALL A00AAF
```

```
      N = 14
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL, (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (' IFAIL=',I4,8(/I4,3D16.7))
      END
*
      SUBROUTINE EX11
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,31), TEMP(30), W(124), ZR(2,30)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
      EXTERNAL         C02AFF
C     .. Data statements ..
C     ==============================
C     RICHARD MARTIN - PROBLEM NO. 2
C     ==============================
      DATA             A(1,1), A(2,1)/1.0D0, 0.0D0/
      DATA             A(1,2), A(2,2)/-0.156071D0, -0.279968D0/
      DATA             A(1,3), A(2,3)/0.047664D0, 0.029149D0/
      DATA             A(1,4), A(2,4)/0.029474D0, -0.033720D0/
      DATA             A(1,5), A(2,5)/0.036454D0, -0.063001D0/
      DATA             A(1,6), A(2,6)/0.058110D0, -0.038271D0/
      DATA             A(1,7), A(2,7)/-0.024418D0, -0.095898D0/
      DATA             A(1,8), A(2,8)/-0.040911D0, 0.018213D0/
      DATA             A(1,9), A(2,9)/0.107986D0, -0.012112D0/
      DATA             A(1,10), A(2,10)/-0.006122D0, 0.064808D0/
      DATA             A(1,11), A(2,11)/-0.058241D0, -0.037196D0/
      DATA             A(1,12), A(2,12)/-0.019662D0, 0.169759D0/
      DATA             A(1,13), A(2,13)/0.096323D0, 0.063821D0/
      DATA             A(1,14), A(2,14)/0.035896D0, -0.082558D0/
      DATA             A(1,15), A(2,15)/0.000140D0, -0.011630D0/
      DATA             A(1,16), A(2,16)/-0.060371D0, -0.034804D0/
      DATA             A(1,17), A(2,17)/0.052805D0, -0.021240D0/
      DATA             A(1,18), A(2,18)/-0.050982D0, 0.030380D0/
      DATA             A(1,19), A(2,19)/0.041200D0, -0.006258D0/
      DATA             A(1,20), A(2,20)/0.015683D0, 0.047790D0/
      DATA             A(1,21), A(2,21)/-0.013689D0, 0.003451D0/
      DATA             A(1,22), A(2,22)/-0.023622D0, -0.001976D0/
      DATA             A(1,23), A(2,23)/-0.011090D0, -0.061840D0/
      DATA             A(1,24), A(2,24)/-0.015088D0, -0.017227D0/
      DATA             A(1,25), A(2,25)/-0.037730D0, -0.056518D0/
      DATA             A(1,26), A(2,26)/-0.011623D0, 0.019046D0/
      DATA             A(1,27), A(2,27)/0.074670D0, -0.059399D0/
      DATA             A(1,28), A(2,28)/0.026980D0, -0.064669D0/
      DATA             A(1,29), A(2,29)/0.003525D0, -0.004874D0/
      DATA             A(1,30), A(2,30)/0.002332D0, 0.035928D0/
      DATA             A(1,31), A(2,31)/0.038520D0, 0.027903D0/
C     .. Executable Statements ..
*     CALL A00AAF
      N = 30
```

```
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL, (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (' IFAIL=',I4,8(/I4,3D16.7))
      END
*
      SUBROUTINE EX12
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 1
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -1.8666666746139526D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 9.9260125411118432D-12
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX13
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 2
C     ================================
```

```
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -1.9333332777023315D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.0280512649421447D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX14
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 3
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -2.0000000794728597D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.0635013814224710D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
```

```
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX15
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 4
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -2.0666666825612386D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.0989513922534314D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX16
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 5
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -2.1333332856496175D-03
      A(1,2) = -3.0533967190711987D-07
```

```
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.1344014030843918D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX17
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 6
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -2.2000000874201457D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.1698515195647182D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX18
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
```

```
      DOUBLE PRECISION A(2,0:3), TEMP(4), W(16), ZR(2,3)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     REBECCA WOODGATE - PROBLEM NO. 7
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = 1.0D0
      A(2,0) = 0.0D0
      A(1,1) = 0.0D0
      A(2,1) = -2.2666666905085246D-03
      A(1,2) = -3.0533967190711987D-07
      A(2,2) = 0.0D0
      A(1,3) = 0.0D0
      A(2,3) = 1.2053015303956785D-11
      N = 3
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
*
      SUBROUTINE EX19
C     .. Local Scalars ..
      INTEGER          I, IFAIL, N
C     .. Local Arrays ..
      DOUBLE PRECISION A(2,0:6), TEMP(7), W(28), ZR(2,6)
C     .. External Functions ..
      DOUBLE PRECISION A02ABF
      EXTERNAL         A02ABF
C     .. External Subroutines ..
C     ================================
C     MICK PONT - PROBLEM NO. 1
C     ================================
      EXTERNAL         C02AFF
C     .. Executable Statements ..
      A(1,0) = -0.398190494797642726E-03
      A(2,0) = -0.161643423352597184E-07
      A(1,1) = -0.630376995875508411E-06
      A(2,1) = 0.679099861801233350E-02
      A(1,2) = 67.8310871970592331
      A(2,2) = 165.126753751424388
      A(1,3) = 2817.40134554712040
      A(2,3) = 7138.34496623663836
      A(1,4) = 414240453.644594848
      A(2,4) = -163541673.279697359
      A(1,5) = -942898.168914408656
      A(2,5) = -2198324.83847518219
```

```
      A(1,6) = -12567222073.7342663
      A(2,6) = 4961096774.84005356
      N = 6
      IFAIL = -1
      CALL C02AFF(A,N,.TRUE.,ZR,W,IFAIL)
      DO 20 I = 1, N
         TEMP(I) = A02ABF(ZR(1,I),ZR(2,I))
   20 CONTINUE
      WRITE (6,FMT=99999) IFAIL
      WRITE (6,FMT=99998)
      WRITE (6,FMT=99997) (I,ZR(1,I),ZR(2,I),TEMP(I),I=1,N)
C
99999 FORMAT (/' C02AFF terminated with IFAIL = ',I4)
99998 FORMAT (/'         real part       imag part        modulus')
99997 FORMAT (8(/I4,3D16.7))
      END
```

B.2 NAG Zerofinder Test and Expected Results

Tests similar to those described in section 3.3 used to test NAG zerofinders.

B.2.1 Expected Results

	Zeros	**Degree**
01	$6.70088 - 7.87599i, 39.7767 + 42.99567i, -7.47753 + 6.88032i$	3
02	$1 + 5i, 2 + 6i, 3 + 7i, 4 + 8i$	4
03	$127.38667077303 + 132.27820320006i, 7.07331324882 - 9.55838903704i,$ $-9.45998402189 + 7.28018583692i, 0, 0$	5
04	$4.16174868 + 3.13751356i, 5.43644837 - 3.97142582i, 2.38988759$ $+7.26807071i, -1.93520144 - 3.97509382i, -2.44755082 + 0.437126175i,$ $-5.27950616 - 2.27596303i, 1.03205812 + 9.29413278i, -4.96687009$ $-8.08712475i, 8.81130928 + 1.54938266i, 10.7976764 + 8.62338151i$	10
05	$0.5 \pm 0.5i, -0.5 - 0.5i, 0.3, -2, i, -0.7i, -0.7i$	8
06	$0.0001i, i, 10000i$	3
07	$2^{-k}(1 + i)$, for $k = 0, 1, \ldots, 9$	10
08	$4i, 3, 3, 2i, 2i, 2i, 1, 1, 1, 1$	10
09	$i, 3i, 1 + 2i, -1 + 2i, -0.5 + 2.866025i, -0.5 + 2.866025i, -0.866025 + 2.5i,$ $0.866025 + 2.5i, -0.866025 + 1.5i, 0.866025 + 1.5i, -0.5 + 1.133974i, 0.5 + 1.133974i$	12
10	$i, -1, 2i, -2, 3i, -3, 4i, -4, 5i, -5$	10
11	$i, 2, 3i, 4, 5i, 6, 7i, 8, 9i, 10$	10
12	$2i, 1 + i, 2, -4i, 2 + 2i, 4, 9i, -3 - 3i, -9, -4 + 4i, -16i, 16, 5 - 5i, 25i, -25$	15
13	$1 + i, 2, -2 + 2i, 4, 3 - 3i, -9, 4 + 4i, -16, -5 - 5i, 25$	10
14	$1, 1, 1, i, i, i$	6
15	$1 + i, 1 + i, 1 + i, -1 + i, -1 + i, -1 + i, 2i, -2i, 2, -2$	10
16	$i, -2, 4i, -8, 16i, -32, 64i, -128, 256i, -512$	10
17	$i, \pm 1, 2i, \pm 2, 3i, \pm 3, 4i, \pm 4, 5i, \pm 5, 6i, \pm 6, 7i, \pm 7, 8i, \pm 8$	24
18	$100 + 101i, 101 + 100i, 99 + 100i, 100 + 99i$	4
19	$1000 + 1001i, 1001 + 1000i, 999 + 1000i, 1000 + 999i$	4
20	$10000 + 10001i, 10001 + 10000i, 9999 + 10000i, 10000 + 9999i$	4

B.2.2 Results obtained

These are the results we obtained with (the modified) C02AFF.

```
 Computed zeros of problem   1
 Z =   3.9776654832D+01 +4.2995667678D+01*i
 Z =   6.7008755006D+00 -7.8759889549D+00*i
 Z =  -7.4775303323D+00 +6.8803212771D+00*i
```

```
Computed zeros of problem   2
Z =   4.0000000000D+00 +8.0000000000D+00*i
Z =   3.0000000000D+00 +7.0000000000D+00*i
Z =   2.0000000000D+00 +6.0000000000D+00*i
Z =   1.0000000000D+00 +5.0000000000D+00*i
Computed zeros of problem   3
Z =   1.2738667077D+02 +1.3227820320D+02*i
Z =  -9.4599840219D+00 +7.2801858369D+00*i
Z =   7.0733132488D+00 -9.5583890370D+00*i
Z =   0.0000000000D+00 +0.0000000000D+00*i
Z =   0.0000000000D+00 +0.0000000000D+00*i
Computed zeros of problem   4
Z =   1.0797676451D+01 +8.6233815278D+00*i
Z =  -4.9668700867D+00 -8.0871247716D+00*i
Z =   1.0320581252D+00 +9.2941327979D+00*i
Z =   5.4364483814D+00 -3.9714258232D+00*i
Z =  -5.2795061629D+00 -2.2759630350D+00*i
Z =   8.8113092846D+00 +1.5493826626D+00*i
Z =   2.3898875901D+00 +7.2680707177D+00*i
Z =  -1.9352014504D+00 -3.9750938123D+00*i
Z =   4.1617486920D+00 +3.1375135599D+00*i
Z =  -2.4475508238D+00 +4.3712617612D-01*i
Computed zeros of problem   5
Z =  -2.0000000000D+00 +5.5511151231D-17*i
Z =   8.3266726847D-17 +1.0000000000D+00*i
Z =   5.0000000000D-01 -5.0000000000D-01*i
Z =  -5.0000000000D-01 -5.0000000000D-01*i
Z =   5.0000000000D-01 +5.0000000000D-01*i
Z =  -9.3900718813D-09 -7.0000004368D-01*i
Z =   9.3900745403D-09 -6.9999995632D-01*i
Z =   3.0000000000D-01 -5.6303759003D-17*i
Computed zeros of problem   6
Z =   0.0000000000D+00 +1.0000000000D+04*i
Z =   0.0000000000D+00 +1.0000000000D+00*i
Z =   0.0000000000D+00 +1.0000000000D-04*i
Computed zeros of problem   7
Z =   1.0000000000D+00 +1.0000000000D+00*i
Z =   5.0000000000D-01 +5.0000000000D-01*i
Z =   2.5000000000D-01 +2.5000000000D-01*i
Z =   1.2500000000D-01 +1.2500000000D-01*i
Z =   6.2500000000D-02 +6.2500000000D-02*i
Z =   3.1250000000D-02 +3.1250000000D-02*i
Z =   1.5625000000D-02 +1.5625000000D-02*i
Z =   7.8125000000D-03 +7.8125000000D-03*i
Z =   3.9062500000D-03 +3.9062500000D-03*i
Z =   1.9531250000D-03 +1.9531250000D-03*i
Computed zeros of problem   8
Z =   1.1324274851D-14 +4.0000000000D+00*i
Z =   3.0000001960D+00 +1.1368287073D-07*i
Z =   2.9999998040D+00 -1.1368287765D-07*i
Z =   5.3703239359D-05 +1.9999957764D+00*i
Z =  -2.3194686893D-05 +2.0000486193D+00*i
Z =  -3.0508552446D-05 +1.9999556043D+00*i
Z =   1.0004206636D+00 +1.8029891200D-04*i
Z =   9.9981958578D-01 +4.2063962886D-04*i
Z =   1.0001802750D+00 -4.2054834104D-04*i
Z =   9.9957947564D-01 -1.8039019982D-04*i
Computed zeros of problem   9
Z =   1.7090280302D-09 +2.9999999997D+00*i
```

```
Z =   5.0000000091D-01 +2.8660254024D+00*i
Z =   8.6602540354D-01 +2.4999999986D+00*i
Z =   9.9999999904D-01 +1.9999999993D+00*i
Z =   4.9999999974D-01 +1.1339745966D+00*i
Z =  -5.0000000036D-01 +1.1339745960D+00*i
Z =  -4.9999999857D-01 +2.8660254048D+00*i
Z =  -8.6602540356D-01 +2.5000000015D+00*i
Z =   8.6602540294D-01 +1.5000000001D+00*i
Z =  -8.6602540464D-01 +1.5000000001D+00*i
Z =  -1.0000000008D+00 +2.0000000010D+00*i
Z =   0.0000000000D+00 +1.0000000000D+00*i
Computed zeros of problem   10
Z =  -5.0000000000D+00 -2.1760371283D-14*i
Z =   3.2418512319D-14 +5.0000000000D+00*i
Z =  -4.9521716856D-14 +4.0000000000D+00*i
Z =  -4.0000000000D+00 +1.2486606750D-15*i
Z =   1.0972541550D-14 +3.0000000000D+00*i
Z =  -3.0000000000D+00 +7.4178411136D-15*i
Z =  -2.0000000000D+00 +1.9644054850D-14*i
Z =  -6.5452075484D-15 +2.0000000000D+00*i
Z =  -1.0000000000D+00 -3.5959920331D-15*i
Z =   8.3054455040D-16 +1.0000000000D+00*i
Computed zeros of problem   11
Z =   1.0000000000D+01 +4.5297099405D-14*i
Z =  -1.4654943925D-14 +9.0000000000D+00*i
Z =   8.0000000000D+00 -1.3756819988D-13*i
Z =  -2.0822796674D-14 +7.0000000000D+00*i
Z =   6.0000000000D+00 -1.2194275024D-14*i
Z =  -4.7942654422D-14 +5.0000000000D+00*i
Z =   4.0000000000D+00 +1.8113580580D-13*i
Z =   1.4949395447D-13 +3.0000000000D+00*i
Z =   2.0000000000D+00 -1.6477410273D-15*i
Z =  -1.3505218387D-16 +1.0000000000D+00*i
Computed zeros of problem   12
Z =  -3.5527136788D-15 +2.5000000000D+01*i
Z =  -2.5000000000D+01 -9.9475983006D-16*i
Z =   1.6000000000D+01 -7.7502705958D-16*i
Z =   3.4117693022D-16 -1.6000000000D+01*i
Z =  -2.4635877038D-15 +9.0000000000D+00*i
Z =  -9.0000000000D+00 +1.6366603938D-15*i
Z =   5.0000000000D+00 -5.0000000000D+00*i
Z =  -4.0000000000D+00 +4.0000000000D+00*i
Z =   4.0000000000D+00 -1.2197473493D-15*i
Z =   4.4482002307D-16 -4.0000000000D+00*i
Z =  -3.0000000000D+00 -3.0000000000D+00*i
Z =   2.0000000000D+00 -1.6930706308D-16*i
Z =   2.0850181768D-16 +2.0000000000D+00*i
Z =   2.0000000000D+00 +2.0000000000D+00*i
Z =   1.0000000000D+00 +1.0000000000D+00*i
Computed zeros of problem   13
Z =   2.5000000000D+01 +2.1726602663D-16*i
Z =  -1.6000000000D+01 -5.9418207868D-16*i
Z =  -9.0000000000D+00 +8.3705218294D-15*i
Z =   4.0000000000D+00 +4.0000000000D+00*i
Z =  -5.0000000000D+00 -5.0000000000D+00*i
Z =   3.0000000000D+00 -3.0000000000D+00*i
Z =   4.0000000000D+00 -4.9878585949D-16*i
Z =  -2.0000000000D+00 +2.0000000000D+00*i
Z =   2.0000000000D+00 +6.0817002009D-17*i
```

```
Z =   1.0000000000D+00 +1.0000000000D+00*i
Computed zeros of problem   14
Z =   1.0000209421D+00 -5.5558006137D-06*i
Z =  -3.3496004290D-06 +1.0000134206D+00*i
Z =   9.9999434034D-01 +2.0914577946D-05*i
Z =   1.3297250819D-05 +9.9999619060D-01*i
Z =   9.9998471759D-01 -1.5358777325D-05*i
Z =  -9.9476503968D-06 +9.9999038881D-01*i
Computed zeros of problem   15
Z =   2.0000000000D+00 +7.7715611724D-16*i
Z =  -1.1102230246D-16 -2.0000000000D+00*i
Z =  -2.0000000000D+00 +1.8185296412D-15*i
Z =   7.1339486510D-15 +2.0000000000D+00*i
Z =   9.9999264604D-01 +1.0000105499D+00*i
Z =  -9.9997868350D-01 +1.0000137728D+00*i
Z =  -1.0000225852D+00 +1.0000115742D+00*i
Z =   1.0000128137D+00 +1.0000010942D+00*i
Z =   9.9999454029D-01 +9.9998835592D-01*i
Z =  -9.9999873126D-01 +9.9997465302D-01*i
Computed zeros of problem   16
Z =  -5.1200000000D+02 +2.8421709430D-14*i
Z =  -2.8421709430D-14 +2.5600000000D+02*i
Z =  -1.2800000000D+02 +9.3942215118D-15*i
Z =  -4.4480689609D-15 +6.4000000000D+01*i
Z =  -3.2000000000D+01 +3.4530397297D-15*i
Z =  -8.6393762760D-16 +1.6000000000D+01*i
Z =  -8.0000000000D+00 -5.7665043499D-17*i
Z =   2.9863256005D-16 +4.0000000000D+00*i
Z =  -2.0000000000D+00 -1.8691279242D-16*i
Z =  -2.8348317857D-17 +1.0000000000D+00*i
Computed zeros of problem   17
Z =   8.0000000000D+00 -1.5676349108D-13*i
Z =  -1.7363888105D-13 +8.0000000000D+00*i
Z =  -8.0000000000D+00 -1.4299221387D-13*i
Z =  -7.1729446375D-13 +7.0000000000D+00*i
Z =   7.0000000000D+00 -9.8828931999D-14*i
Z =  -7.0000000000D+00 -3.2178108275D-15*i
Z =   9.3174285646D-13 +4.0000000000D+00*i
Z =   6.0000000000D+00 +1.0448247645D-12*i
Z =   2.9547737248D-12 +6.0000000000D+00*i
Z =  -2.8702458975D-12 +5.0000000000D+00*i
Z =  -6.0000000000D+00 +9.8278667546D-13*i
Z =  -5.0000000000D+00 -1.1361775741D-12*i
Z =   5.0000000000D+00 -8.1251528976D-13*i
Z =  -4.0000000000D+00 +1.3530093658D-13*i
Z =   4.0000000000D+00 -2.2188333578D-13*i
Z =  -3.0000000000D+00 -2.3829736724D-15*i
Z =   3.0000000000D+00 +4.3671772351D-14*i
Z =  -2.0000000000D+00 -2.0059639466D-15*i
Z =  -1.3402009401D-15 +1.0000000000D+00*i
Z =   2.0000000000D+00 -3.0925145376D-15*i
Z =  -2.2129652046D-13 +3.0000000000D+00*i
Z =   2.4236870952D-14 +2.0000000000D+00*i
Z =   1.0000000000D+00 -2.2174354616D-16*i
Z =  -1.0000000000D+00 +7.2260592489D-17*i
Computed zeros of problem   18
Z =   1.0000000003D+02 +1.0100000001D+02*i
Z =   1.0100000001D+02 +9.9999999969D+01*i
Z =   9.8999999993D+01 +1.0000000003D+02*i
```

```
Z =   9.9999999969D+01 +9.8999999993D+01*i
Computed zeros of problem   19
Z =   1.0000000978D+03 +1.0000012661D+03*i
Z =   1.0000001451D+03 +1.0000011776D+03*i
Z =   9.9999815037D+02 +9.9999894947D+02*i
Z =   1.0000016067D+03 +9.9999860675D+02*i
Computed zeros of problem   20
Z =   1.0000000000D+04 +1.0000000000D+04*i
Z =   1.0000000000D+04 +1.0000000000D+04*i
Z =   1.0000000000D+04 +1.0000000000D+04*i
Z =   1.0000000000D+04 +1.0000000000D+04*i
```

Appendix C

Comparisons of C02AFF with other zerofinders

Here are our codes for computing zeros of random polynomials using various zerofinders. We also present some of the results obtained.

C.1 FORTRAN code for computing 'exact' zeros of random polynomials

```
C     exroots.f
C     COMPUTING THE ROOTS OF A POLYNOMIAL USING COMPANION
C     MATRIX AND TO QUADRUPLE PRECISION (LAPACK)
C
      IMPLICIT NONE
      INTEGER NX,NCOUNT,FOUT,FOUT2,POL
      PARAMETER (NX=10,FOUT=10,FOUT2=12,POL=100)
      DOUBLE PRECISION B(2,0:NX),A1,A2,E1,E2,
     *         RWRK(2*NX)
      INTEGER I,J,N,INFO,K
      DOUBLE COMPLEX A(NX,NX),VL(NX,NX),VR(NX,NX),WRK(3*NX),
     *       EIGS(NX)
      REAL DRAND,X, TX,TIME(2),TTIME, DTIME
      EXTERNAL ZGEEV
C
C     ....Executable statements ....
C
      OPEN(FOUT,FILE='c10fs')
      OPEN(FOUT2,FILE='ze10os')
      N = NX
      NCOUNT=0
      TTIME = 0.D0
      DO 500 K=1,POL
C     Setup random coefficients for degree-10  monic polynomial
C
         X = DRAND(K)
         B(1,0) = 1.D0
         B(2,0) = 0.D0
         DO 100 I=1,NX
            A1 = DBLE(DRAND(0))*2.D0 - 1.D0
            A2 = DBLE(DRAND(0))*2.D0 - 1.D0
            E1 = DBLE(DRAND(0))*20.D0 - 10.D0
            E2 = DBLE(DRAND(0))*20.D0 - 10.D0
```

```
            B(1,I)= A1*10**E1
            B(2,I)= A2*10**E2
 100     CONTINUE
C
C     Save coefficients to compare with MATLAB
C     and C02AFF and whatever.
C
         DO 125 I=1,NX
            WRITE(FOUT,9997)B(1,I)
            WRITE(FOUT,9997)B(2,I)
 125     CONTINUE
C
C     Setup companion matrix for polynomials
C
         DO 200 I=1,N
            DO 300 J=1,N
               IF (J.EQ.N) THEN
                  A(I,J) = -DCMPLX(B(1,N-I+1),B(2,N-I+1))
               ELSE IF (J.EQ.I-1) THEN
                  A(I,J) = DCMPLX(1.d0,0.d0)
               ELSE
                  A(I,J) = DCMPLX(0.d0,0.d0)
               END IF
 300        CONTINUE
 200     CONTINUE
C
C     Calculate eigenvalues of companion matrix (LAPACK)
C     and calculate time taken to compute eigenvalues
         TX = DTIME(TIME)
         CALL ZGEEV('N','N',N,A,N,EIGS,VL,N,VR,N,WRK,3*N,RWRK,INFO)
         TTIME = TTIME + DTIME(TIME)
         NCOUNT = NCOUNT + 1
C
C     Print eigenvalues of companion matrix (roots) to file
C
            WRITE(FOUT2,9998)(EIGS(I),I=1,N)
 500  CONTINUE
WRITE(*,*)'NCOUNT=',NCOUNT
       WRITE(*,*)'TIME=',TTIME
C
 9997 FORMAT(D36.30)
 9998 FORMAT(D36.30 / D36.30)
 9999 FORMAT(2D36.30)
      CLOSE(FOUT)
      CLOSE(FOUT2)
      STOP
      END
```

C.2 Computing zeros with C02AFF

This is our code to compute zeros using C02AFF for the same random polynomials created using EXROOTS. Note that the codes for PA16, CPOLY and ZROOTS is quite similar and we shall therefore not present them.

```
C     This reads coefficients of random polynomials, and tries out C02AFF
C     on them. Saves results in another file for comparison with MATLAB
      IMPLICIT NONE
      INTEGER NX,NCOUNT,FIN,FOUT,FOUT2,POL
      PARAMETER (NX=10,FIN=10,FOUT=12,POL=100)
```

```
      DOUBLE PRECISION ACO(2,NX+1),Z(2,NX),W(4*(NX+1)),DTIME,TTIME
      INTEGER I,N,IFAIL,IGO,J
      REAL TX,TIME(2)
      EXTERNAL CO2AFF
C
C     ....Executable statements ....
C
      OPEN(FIN,FILE='c10fs')
      OPEN(FOUT,FILE='fp10')
      N = NX
      NCOUNT=0
      DO 300 IGO=1,POL
         ACO(1,1)=1.D0
         ACO(2,1)=0.D0
C     Read in coefficients from 'c10fs'
C
         DO 77 I=2,N+1
            READ(FIN,*)ACO(1,I)
            READ(FIN,*)ACO(2,I)
 77      CONTINUE
         IFAIL=-1
         TX = DTIME(TIME)
         CALL CO2AFF(ACO,N,.TRUE.,Z,W,IFAIL)
         TTIME = TTIME + DTIME(TIME)
         NCOUNT = NCOUNT + 1
         WRITE(FOUT,120)(Z(1,J),Z(2,J),J=1,N)
 100     FORMAT(2I8,2F12.5)
 120     FORMAT(D28.20,3X,D28.20)
 300  CONTINUE
      WRITE(*,*) 'NCOUNT=',NCOUNT
      WRITE(*,*) 'TIME = ',TTIME
      CLOSE(FIN)
      CLOSE(FOUT)
      STOP
      END
```

C.3 Computing zeros with MATLAB

This is our code to compute zeros using MATLAB's `roots`, for the same random polynomials created using EXROOTS.

```
% rts10.m -
% This program calculates the roots of 100 random
% degree 10 polynomials, reading coefficients
% from c10fs using roots
clear
format long
mtime = 0;
pol = 100; % Number of polynomials
deg = 10;  % Degree of each polynomial
num = pol*deg; % Total number of zeros
n = deg-1;  % step
load c10fs;
z = c10fs(:);
x = z(1:2:end);
y = z(2:2:end);
cofs = x + i*y;
fid = fopen('mp10','w');
```

```
for i=1:deg:num
   p1 = cofs(i:i+n);
   p = [1;p1];
   tic
   zer = roots(p);
   mtime = mtime + toc;
   for k=1:deg
      fprintf(fid,'%28.20f  %28.20f\n',real(zer(k)),imag(zer(k)));
   end
end
mtime
fclose(fid);
```

C.4 Calculating error and Scatter plots

This is the code used to calculate the error in `roots` and C02AFF and produce a scatter plot. The code for the other zerofinders requires just a few appropriate changes.

```
% error1.m -- Produce scatter plot for the error in
% roots and C02AFF
%
% Load computed zeros of all cases
pol = 100; % Number of polynomials
deg = 10;  % degree of polynomial
num = pol*deg; % Total number of zeros
n = deg-1;  % step
load ze10os;
load mp10;
load fp10;
xzer = ze10os(1:2:end);
yzer = ze10os(2:2:end);
zstd = xzer + i*yzer;
xm = mp10(:,1);
ym = mp10(:,2);
zm = xm + i*ym;
xc = fp10(:,1);
yc = fp10(:,2);
zc = xc + i*yc;
% sort zeros for each polynomial for comparison
for ii = 1:deg:num
   kd = zstd(ii:ii+n);
   [jk,in] = sort(abs(kd));
   zstd(ii:ii+n) = kd(in);
   km = zm(ii:ii+n);
   [jk,in] = sort(abs(km));
   zm(ii:ii+n) = km(in);
   kc = zc(ii:ii+n);
   [jk,in] = sort(abs(kc));
   zc(ii:ii+n) = kc(in);
end
%Compute errors
%
errc02 = abs(zstd - zc);
errmat = abs(zstd - zm);
% scatter plots
figure
loglog(errc02,errmat,'x')
hold on
```

```
plot([1e-30 1],[1e-30 1],'r')
xlabel('error in C02AFF')
ylabel('error in ROOTS')
title([' DEGREE=' int2str(deg) ' No RANDOM POLYNOMIALS =' int2str(pol)])
grid on
```

C.5 More scatter plots

These are other scatter plots which we obtained on comparing the error in polynomials of other degrees.

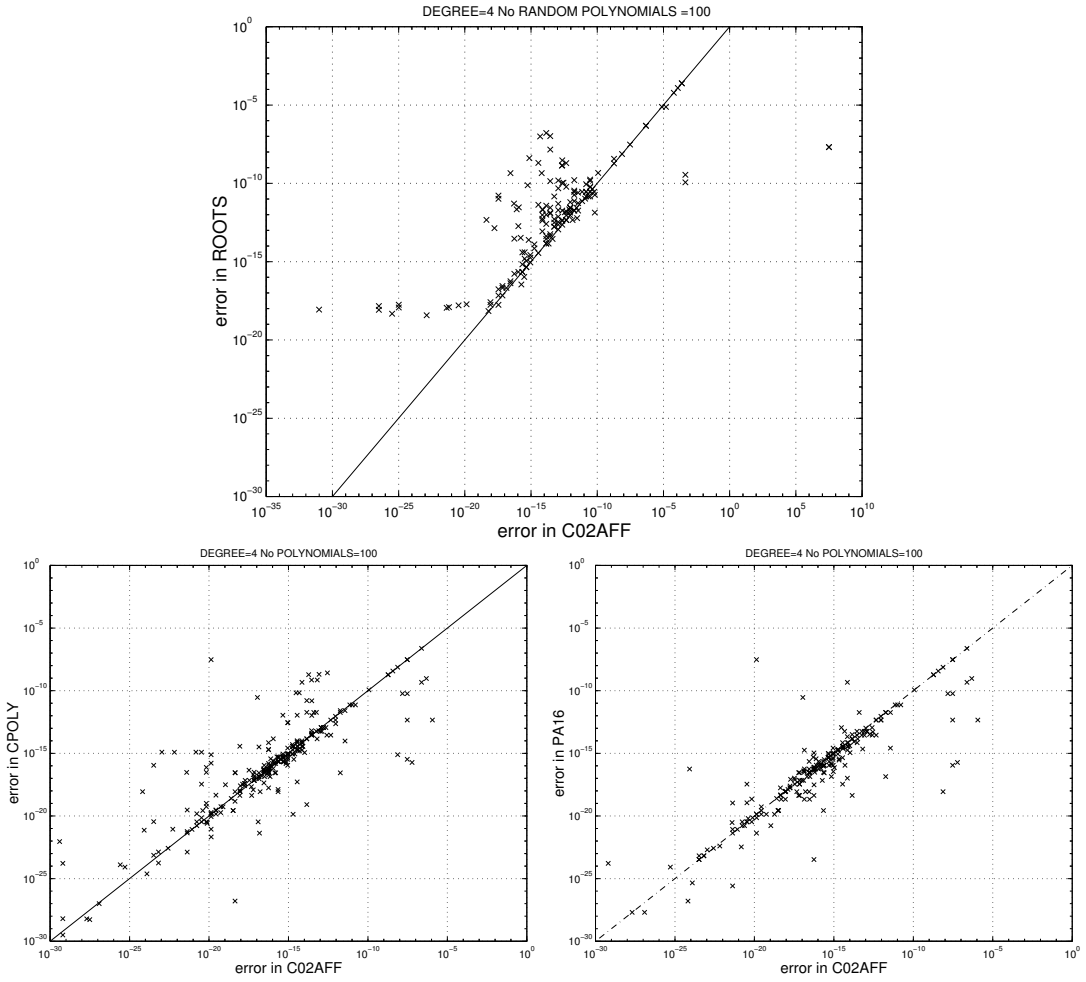


Figure C.1: Scatter plots showing for degree-4 random polynomials

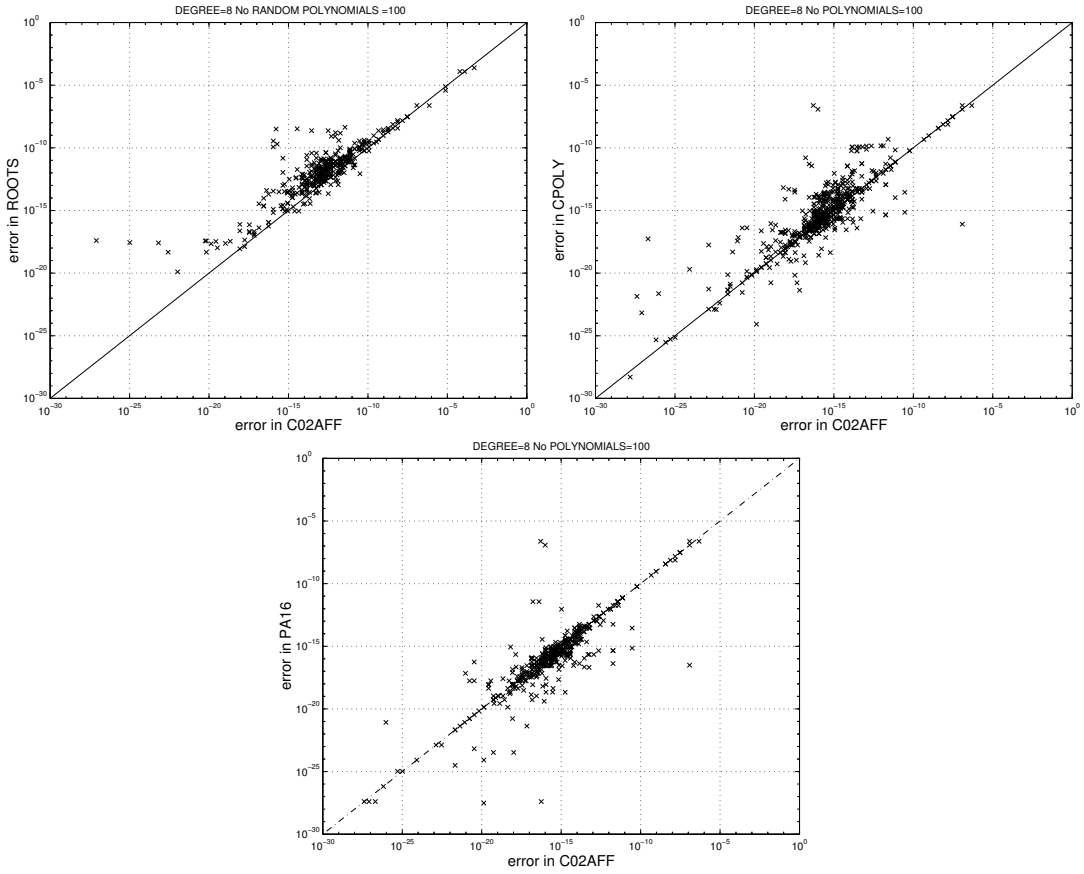


Figure C.2: Scatter plots showing for degree-8 random polynomials

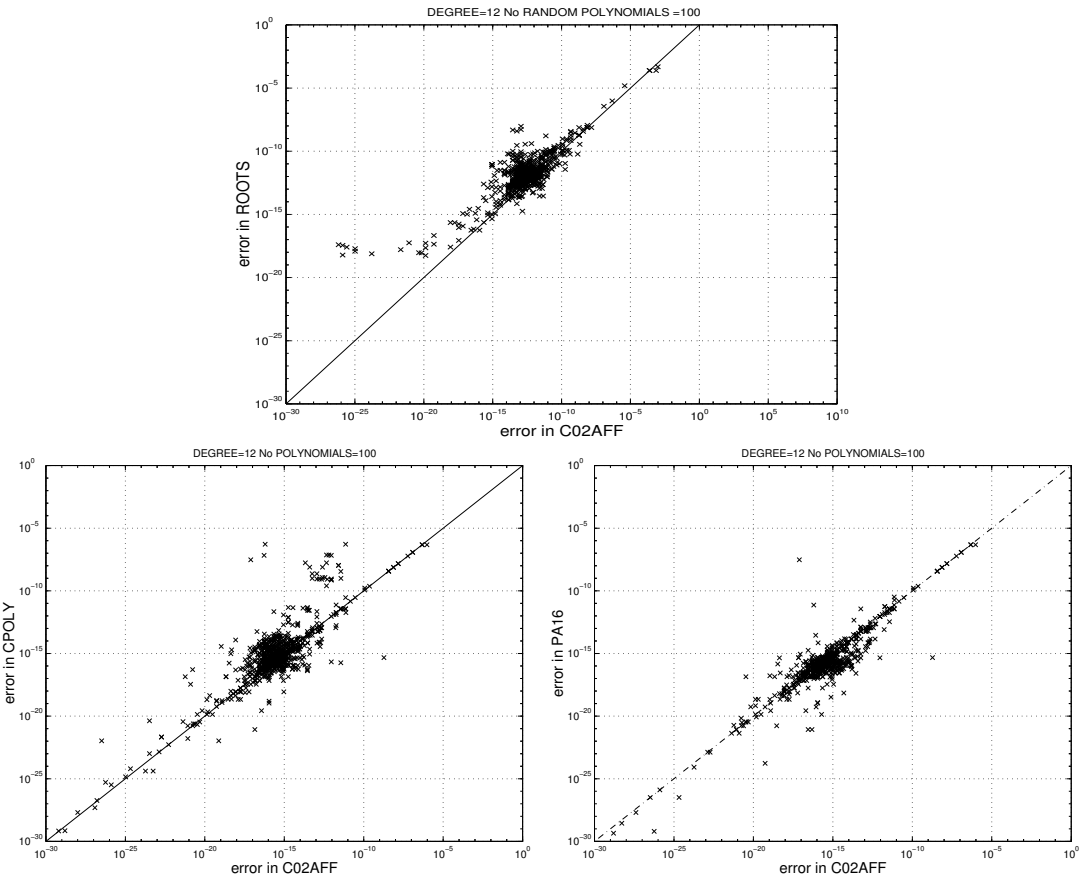


Figure C.3: Scatter plots showing for degree-12 random polynomials

Bibliography

[ABB+99] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greennbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. SIAM, 3rd edition, 1999.

[Art72] D. W. Arthur. Extension of Bairstow's method for multiple quadratic factors. *J. Inst. Math. Appl.*, 9:194–197, 1972.

[BM67] A. Balfour and A. J. McTernan. *The Numerical Solution of Equations*. Heimann Educational Books Ltd, 1967.

[Bro75] K. W. Brodlie. On Bairstow's method for the solution of polynomial equations. *Math. Comp.*, 29:816–826, 1975.

[Cop62] E. T. Copson. *An Introduction to the Theory of Functions of a Complex Variable*. Oxford University Press, 1962.

[DB74] G. Dahlquist and Å. Björck. *Numerical Methods*. Prentice-Hall, 1974.

[EM95] A. Edelman and H. Murakami. Polynomial roots from companion matrix eigenvalues. *Math. Comp*, 64(210):763–776, 1995.

[Frö85] Carl-Erik Fröberg. *Numerical Mathematics*. Benjamin/Cummings, 1985.

[Gau97] W. Gautschi. *Numerical Analysis: An Introduction*. Birkhäuser, 1997.

[Goe94] S. Goedecker. Remark on algorithms to find roots of polynomials. *SIAM J. Sci. Comp*, 15(5):1059–1063, September 1994.

[GR67] G. H. Golub and T. N. Robertson. A generalized Bairstow algorithm. *Comm. ACM*, 10:371–373, 1967.

[Hen64] P. Henrici. *Essentials of Numerical Analysis*. Wiley and Sons, 1964.

[Hen74] P. Henrici. *Applied and Computational Complex Analysis*, volume 1. Wiley and Sons, 1974.

[Hen82] P. Henrici. *Essentials of Numerical Analysis with Pocket Calculator Demonstrations*. Wiley and Sons, 1982.

[HP77] E. Hansen and M. Patrick. A family of root-finding methods. *Numer. Math.*, 27:257–269, 1977.

[HPR77] E. Hansen, M. Patrick, and J. Rusnak. Some modifications of Laguerre's method. *BIT*, 17:408–417, 1977.

[JT70a] M. A. Jenkins and J. F. Traub. A three-stage algorithm for real polynomials using quadratic iteration. *SIAM Journal on Numerical Analysis*, 7:545–566, 1970.

[JT70b] M. A. Jenkins and J. F. Traub. A three-stage variable-shift iteration for polynomial zeros in relation to generalized Rayleigh iteration. *Numer. Math.*, 14:252–263, 1970.

[JT74] M. A. Jenkins and J. F. Traub. Principles for testing polynomial zerofinding programs. Technical report, Carnegie-Mellon University, March 1974.

[Leh63] D. H. Lehmer. The complete root-squaring method. *J. Soc. Indust. Appl. Math.*, 11:705–717, 1963.

[LF94] Markus Lang and Bernhard-Christian Frenzel. A fast and efficient program for finding all polynomial roots. In *Proc. ISCAS*, 1994.

[McN93] J. M. McNamee. A bibliography on roots of polynomials. *J. Comp. Appl. Math.*, 47:391–394, 1993. Online access to updated bibliography is available at `http://www.elsevier.nl/homapage/sac/cam/mcnamee/.`

[Mol91] C. B. Moler. ROOTS–Of polynomials, that is. *The MathWorks Newsletter*, 5(1):8–9, 1991.

[Ost60] A. M. Ostrowski. *Solution of Equations and System of Equations.* Academic Press, 1960.

[Ost73] A. M. Ostrowski. *Solution of Equations in Euclidean and Banach Spaces.* Academic Press, 1973.

[Pan97] V. Pan. Solving a polynomial equation: Some history and recent progress. *SIAM Review*, 39(2):187–220, June 1997.

[Par63] B. Parlett. Laguerre's method applied to the matrix eigenvalue problem. *Math. Comp*, 18:464–485, 1963.

[PFTV86] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes.* Cambridge University Press, 1986. (FORTRAN 77 edition).

[RR78] A. Ralston and P. Rabinowitz. *A First Course in Numerical Analysis.* McGraw-Hill, 1978.

[Smi67] B. T. Smith. ZERPOL, A zerofinding algorithm for polynomials using Laguerre's method. Technical report, University of Toronto, May 1967.

[Tra64] J. F. Traub. *Iterative Methods for the Solution of Equations.* Prentice-Hall, 1964.

[Tre01] L. N. Trefethen. Companion matrices and polynomial zeros. (Unpublished draft section of book to appear), July 2001.

[TT94] K. Toh and L. N. Trefethen. Pseudozeros of polynomials and pseudospectra of companion matrices. *Numer. Math.*, (68):403–425, 1994.

[Ueb95] C. W. Ueberhuber. *Numerical Computation 2.* Springer, 1995.

[Wil63] J. H. Wilkinson. *Rounding Errors in Algebraic Processes.* Her Majesty's Stationary Office, 1963.

[Wil65] J. H. Wilkinson. *The Algebraic Eigenvalue Problem.* Oxford University Press, 1965.