Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Physics Communications 175 (2006) 304–313
Computer Physics Communications
www.elsevier.com/locate/cpc

Computing zeros of analytic functions in the complex plane without using derivatives ☆

C.J. Gillan *, A. Schuchinsky, I. Spence

The Institute for Electronics, Communications and Information Technology (ECIT), The Queen's University of Belfast, The Northern Ireland Science Park, Queen's Road, Queen's Island, Belfast, BT3 9DT, UK

Received 5 May 2005; received in revised form 14 February 2006; accepted 29 April 2006

Available online 23 June 2006

Abstract

We present a package in Fortran 90 which solves $f(z) = 0$, where $z \in \mathbb{W} \subset \mathbb{C}$ without requiring the evaluation of derivatives, $f'(z)$. $\mathbb{W}$ is bounded by a simple closed curve and $f(z)$ must be holomorphic within $\mathbb{W}$.

We have developed and tested the package to support our work in the modeling of high frequency and optical wave guiding and resonant structures. The respective eigenvalue problems are particularly challenging because they require the high precision computation of all multiple complex roots of $f(z)$ confined to the specified finite domain. Generally $f(z)$, despite being holomorphic, does not have explicit analytical form thereby inhibiting evaluation of its derivatives.

Program summary

Title of program: EZERO
Catalogue identifier: ADXY_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/ADXY_v1_0
Program obtainable from: CPC Program Library, Queen's University of Belfast, N. Ireland
Computer: IBM compatible desktop PC
Operating system: Fedora Core 2 Linux (with 2.6.5 kernel)
Programming languages used: Fortran 90
No. of bits in a word: 32
No. of processors used: one
Has the code been vectorized: no
No. of lines in distributed program, including test data, etc.: 21045
Number of bytes in distributed program including test data, etc.: 223 756
Distribution format: tar.gz
Peripherals used: none
Method of solution: Our package uses the principle of the argument to count the number of zeros encompassed by a contour and then computes estimates for the zeros. Refined results for each zero are obtained by application of the derivative-free Halley method with or without Aitken acceleration, as the user wishes.

PACS: 02.30.Dk; 02.60.Cb; 02.70.Pt; 02.70.-c

Keywords: Analytic functions; Zeros; Computation of zeros; Halley's method

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author.
E-mail address: c.gillan@ecit.qub.ac.uk (C.J. Gillan).

0010-4655/$ – see front matter
doi:10.1016/j.cpc.2006.04.007

1. Introduction

In several fields of computational science one is required to obtain numerical solutions to the problem

$$f(z) = 0, \tag{1}$$

where $f(z)$ is an analytic function defined only on a restricted domain $\mathbb{W}$ of the complex plane $\mathbb{C}$, and $\mathbb{W}$ is bounded by a simply connected closed curve, Γ. We always assume that no root of $f(z)$ actually lies on Γ; this is not a severe restriction because the contour can generally be chosen such that its does not pass through a root.

A complicating factor in such kind of problems is usually that, although $f(z)$ is analytic on $\mathbb{W}$, either the function cannot be easily differentiated or it is such that the numerical calculation of its derivatives, in addition to repeated evaluation of the function, is computationally prohibitive. Such a situation is typical for the electromagnetic problems solved by moment methods where $f(z)$ is represented as the determinant of a matrix, each element of which is expressed by integrals or series [1–7]. Similar problems arise in the study of intermediate energy electron scattering by atoms and molecules [8] where z represents the energy of the incident electron. Computation of the resonance parameters for the electron scattering requires the solution of $f(z) = 0$ where $f(z)$ is defined with respect to a matrix. For each value of z the matrix, but not its derivative, may be calculated using large computer program suites [9]. In these circumstances, it is therefore necessary to employ solution methods which do not require derivative evaluations and this paper presents a computer program developed for that purpose. In order to rigorously test and validate our program, we demonstrate the verification results for functions whose roots are already known or can be obtained analytically.

Iterative methods [10] are often employed for the refinement of the numerical solution of (1), i.e. if we have, an approximation z_n for a solution, then we may obtain its better estimate, z_{n+1} by applying the iterative procedure [10]

$$z_{i+1} = \mathbf{H}(z_i), \tag{2}$$

where $\mathbf{H}(z_i)$ is an interpolation function. This process is repeated until we satisfy the condition

$$\left|f(z_{i+1})\right| < \epsilon, \tag{3}$$

where ϵ is a user defined threshold. The distinction between various iteration methods essentially arises in the different forms of the interpolation function $\mathbf{H}$ used.

We have found Halley's method [10,11] to be particularly computationally efficient and adopted it in this paper. Although the standard formulation of Halley's method involves derivatives, the use of finite differences, instead of the derivatives, maintains the same convergence rate [10], and we have implemented this finite difference approach in our work.

The set of values z_n generated by (2) is an iterated map of the complex plane. In order to ensure convergence, the initial estimate z_0 must lie in the basin of attraction of the root with respect to $\mathbf{H}$, that is

$$z_0 \in K_c, \quad z_0 \notin J_c, \tag{4}$$

where K_c is the filled-in Julia Set for $\mathbf{H}$ and J_c its Julia set [12]. However a priori, it is essentially impossible to know whether a given z_0 satisfies (4) or not. Furthermore this can only be determined after executing a sufficiently large number of iterations [13]. Alternatively, one needs a priori knowledge of the function properties and root location when picking z_0. Yau and Ben-Israel [14] have shown that, at least in some instances, the Julia set for Halley iterations has a less chaotic shape that the equivalent for Newton iterations.

In applying an iteration technique directly to the stated problem, no account of the function analyticity is taken at the onset. It is only checked that $z_n \in \mathbb{W}$. Clearly, if after a few iterations it appears that $z_n \notin \mathbb{W}$, one has no choice but to halt the iterations, as there may be no way of converging to a zero of $f(z)$ located inside $\mathbb{W}$. Then the process should be restarted from the beginning with a different initial value of z_0. The occurrence of this situation is a reflection of the fact that in general, the basic iterative methods do not guarantee convergence within the finite domain, especially when $f(z_n)$ possesses multiple roots. It is therefore important that the root isolation step produces suitable values.

Indeed, making use of the Cauchy theorem, function $f(z_i)$ can be expressed, at any internal point $z_i \in \mathbb{W}$, by an integral of $f(z)$ around the contour Γ which encloses $\mathbb{W}$. Previous work in this journal by Botten et al. [15] extended the technique of Delves and Lyness [16] which highlighted the power of using contour integrals to compute the points at which $f(z) = 0$. This work of Botten et al. [15] employed circular contours which are traditionally used in complex analysis. However, when $f(z)$ has branch cuts or $\mathbb{W}$ has complicated shape which should be spanned without gaps and overlaps by several sub-domains, circular contours are hardly applicable. Therefore in this work we use rectangular contours. In particular, this obviates the need to perform co-ordinate transformations to circles, a process which is generally computationally expensive [17].

There is one situation however in which the contour integral method may encounter problems. This happens when the root or, indeed, multiple roots lie very close to the contour, a situation in which the evaluation of the contour integrals is extremely difficult. In some problems, there are additional roots lying nearby but outside the contour, Γ, so the simple idea of perturbing the contour slightly may be inappropriate.

From a practical computational perspective, the contour integral method will only produce estimates, though reasonably good approximations in general, to the exact roots. Limited accuracy of this approach is associated with the inherent errors of numerical quadratures and by the finite dynamic range of the floating-point arithmetic available on the particular computer processors used. Thus, if a computed estimate is such that

$$\left|f(z_i)\right| \geqslant \epsilon, \tag{5}$$

then the iterative methods may be applied to refine the root. Splitting the numerical solution of (1) into two stages is of fundamental significance for reliable evaluation of multiple roots of $f(z)$. Thus, roots are separated or bracketed at first. Then each root estimate is refined independently inside the specified sub-domain of $\mathbb{W}$. We have implemented this two-stage

approach in a Fortran 90 program where the Halley iterative algorithm has been adapted for the root refinement in the restricted domain containing a single root.

In the next section, we present a situation in computational electromagnetics which requires use of our algorithm. Next we present the analytical and computational details of our algorithm. The following section describes the format for inputing data to our Fortran computer program; section four presents some details of the implementation of our code and the following section describes the output from some test runs. The final section of the paper outlines our conclusions and some directions for future work.

2. The dispersion equation for a semiconductor layer

Silicon based integrated micro systems combine multiple active and passive components densely packed in layers of substrates and when operated in the terahertz and gigahertz range, such assemblies cause delay, cross-talk and distortion of transmitted signals. In order to minimize the losses, it is necessary to model and to compute electromagnetic wave propagation in multilayered structures [18].

One model, which is an extension of previous work [18], is illustrated in Fig. 1 where a layer of metal of thickness d is surrounded by two layers of purely dielectric material (typically glass) which in turn are surrounded by perfect conductors enclosing the structure. At the frequencies concerned (approximately 20–100 THz) metals, such as gold, have lossy dielectric semiconductor properties meaning that their relative permittivity ϵ_m is a complex number and is also a function of frequency. Often, this dependence of ϵ_m is available only via a table of numerical values and not as an analytic function. The upper and lower layers are assumed to be pure dielectric material with constant relative permittivities meaning that $\epsilon_1, \epsilon_2 \in \mathbb{R}$.

Considering just transverse magnetic (TM) waves propagating along the z-axis in Fig. 1 and applying the boundary conditions for field continuity at the various layer interaces yields a dispersion relation [7,18] for the metal layer which defines γ_n, a complex number, as a function of frequency f (or wavelength λ in vacuum), in the form

$$\begin{aligned}&\gamma_n^2 + \epsilon_m^2 \frac{\beta_{1n}}{\epsilon_1} \tanh(\beta_{1n} k_0 a_1) \frac{\beta_{2n}}{\epsilon_2} \tanh(\beta_{2n} k_0 a_2) \\ &\quad + \epsilon_m \gamma_n \coth(\gamma_n k_0 d) \\ &\quad \times \left(\frac{\beta_{2n}}{\epsilon_2} \tanh(\beta_{2n} k_0 a_2) + \frac{\beta_{1n}}{\epsilon_1} \tanh(\beta_{1n} k_0 a_1) \right) = 0, \end{aligned} \tag{6}$$

where

$$\beta_{in} = \sqrt{\gamma_n^2 + \epsilon_m - \epsilon_i}, \quad i = 1, 2, \quad \text{and} \quad k_0 = \frac{2\pi}{\lambda}. \tag{7}$$

Eliminating the poles in Eq. (6) results in the generic dispersion equation

$$\begin{aligned}&\frac{\sinh(\gamma_n k_0 d)}{\gamma_n} \left[\gamma_n^2 \cosh(k_0 a_1 \beta_{1n}) \cosh(k_0 a_2 \beta_{2n}) \right. \\ &\quad \left. + \epsilon_m^2 \frac{\beta_{1n}}{\epsilon_1} \sinh(k_0 a_1 \beta_{1n}) \frac{\beta_{2n}}{\epsilon_2} \sinh(k_0 a_2 \beta_{2n}) \right] \\ &\quad + \epsilon_m \cosh(\gamma_n k_0 d) \left[\frac{\beta_{2n}}{\epsilon_2} \sinh(k_0 a_2 \beta_{2n}) \cosh(k_0 a_1 \beta_{1n}) \right. \\ &\quad \left. + \frac{\beta_{1n}}{\epsilon_1} \sinh(k_0 a_1 \beta_{1n}) \cosh(k_0 a_2 \beta_{2n}) \right] = 0. \end{aligned} \tag{8}$$

For any given set of layer thicknesses, Eq. (8) is solved by first fixing λ (or f) which then determines k_0 and ϵ_m. Thus we have

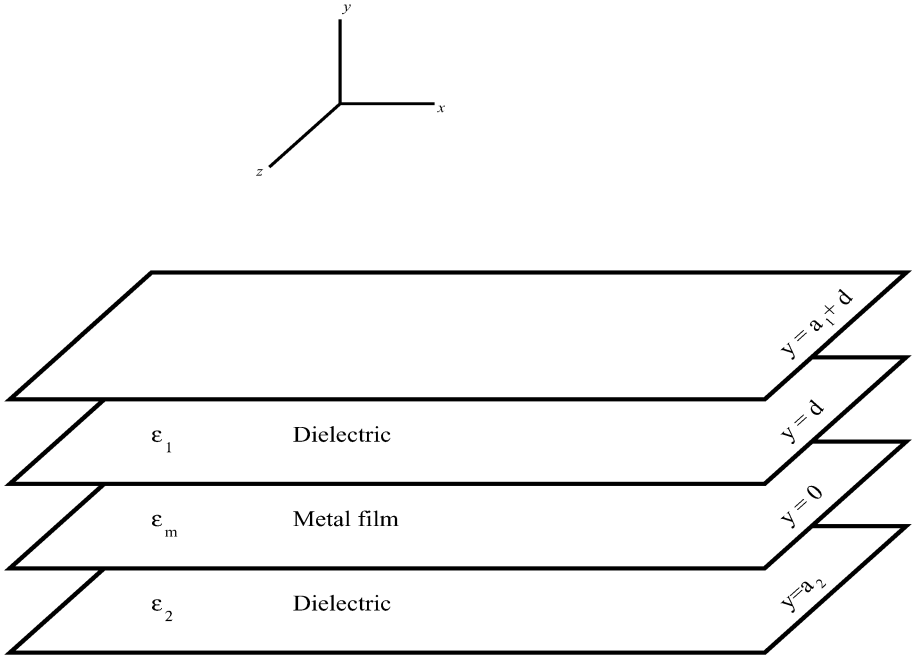

Fig. 1. A layer of metal (gold), thickness d and relative permittivity ϵ_m, in a parallel plate waveguide sandwiched between dielectric layers of thicknesses a_1, a_2 and relative permittivities ϵ_1 and ϵ_2. At $y = a_1 + d$ and $y = a_2$ there are perfect conductors enclosing the structure. All layers are of infinite extent in the xz-plane and have constant relative permeabilities, μ.

the form

$$f(\gamma_n) = 0 \tag{9}$$

and we seek solutions for γ_n over the search domain. The limits on the search domain for γ_n are defined by the physics of the problem.

3. The method

In our work we take $\mathbb{W}$ to be a rectangular region in the complex plane and Γ to be its perimeter. For any analytical function $f(z)$ defined on the rectangle, we implement a multi-step process to solve Eq. (1).

(1) Count the number N_r of roots enclosed by the rectangle.
(2) If N_r is greater than a user defined threshold then sub-divide the region into smaller rectangles, and repeat, until each sub-rectangle has at most N_r roots within. This is similar to the approach taken by Dellnitz et al. [19].
(3) Reformulate the problem of isolating the roots in each sub-rectangle as a generalized eigenvalue problem and solve it numerically for these eigenvalues.
(4) For each root computed, check whether the value of $|f(z)|$ is less than a user defined threshold. If this condition is not fulfilled at the root, then apply Halley's method, using the computed root as a starting point, to hopefully refine the root within the required convergence bound.

3.1. Counting the roots

A fundamental component of our code is the rapid evaluation of the number of roots that lie within the rectangle, $\mathbb{W}$. It is based on the conventional principle of argument [20], which gives the number N_r of $f(z)$ zeros

$$N_r = \frac{1}{2\pi} \int_{\Gamma} \mathrm{d}\big(\arg\big\{f(z)\big\}\big) \tag{10}$$

meaning that the number of zeros encompassed by the contour Γ is merely determined by the $\arg\{f(z)\}$ variation on Γ.

The counting algorithm above has been implemented in subroutine COUNTZ. Starting at the lower left-hand corner of the rectangle, we compute $f(z)$ and then $\arg\{f(z)\}$. Small steps are then taken along the contour and the process repeated in order to monitor the variation in the argument. On returning to the starting point we have an estimate for the number of zeros encompassed by the closed contour Γ.

An example output, which is available through the debug flags in the COUNTZ, from the argument calculation, is shown in Fig. 2 for $f(z) = z$. The function has a root at $z = 0$ and this is reflected in the fact that the argument increases by 2π around the contour.

The size of the steps taken is a key determinant in a successful application of the technique, in part because the computation of the argument employs the function ATAN2 which only returns only its principal value defined in the range $[-\pi, \pi]$. In the case of multiple roots, the computed argument must be offset appropriately by multiples of 2π as we traverse the contour.

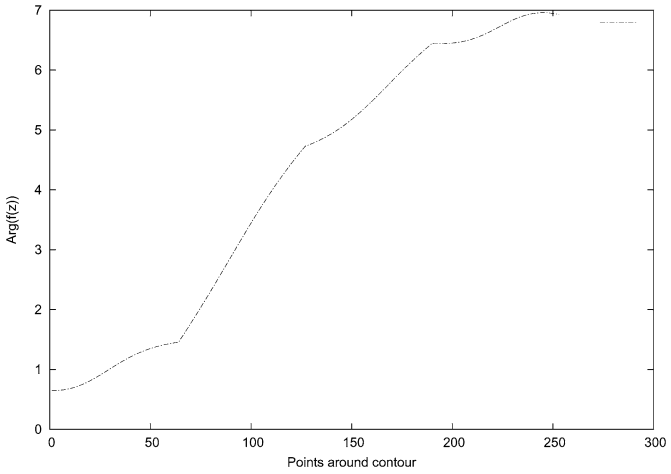

Fig. 2. The argument of $f(z) = z$ for one complete closed loop around the square with vertices $(-1,-1)$, $(1,-1)$, $(1,1)$, $(-1,1)$. The first and last points on the x-axis correspond to $z = (-1,-1)$.

Clearly for rapidly varying functions $f(z)$ the steps taken must be kept sufficiently small so that no Riemann sheets are missed. The size of the steps taken is controlled by the input parameter `npts` described later. If `npts` is too small compared to the variation in the argument of the function around the contour, it is possible to miss a transition through a multiple of 2π and therefore underestimate the number of roots lying within the contour. In the present implementation of COUNTZ we have chosen a fixed number of steps. Each side of the rectangle is divided into 1024 points, although this may be easily varied by changing the parameter *npts* at the top of the routine.

3.2. Localizing the roots

In common with other authors [21–23] we denote the N zeros of $f(z)$, lying within Γ, as $Z_k, k = 1, \ldots, N$. If $f(z)$ has multiple roots then, of course, some of the Z_k will be identical. We may define a symmetric bi-linear form as follows

$$\langle \phi, \psi \rangle = \frac{1}{2\pi \mathrm{i}} \int_{\Gamma} \phi(z)\psi(z)\frac{1}{f(z)}\,\mathrm{d}z, \tag{11}$$

where ϕ and ψ are polynomials. The integrand in (11) has a pole at each Z_k and therefore the integral may be evaluated using the calculus of residues. Using Eq. (11) we define the moments s_p by

$$s_p = \big\langle 1, z^p \big\rangle \tag{12}$$

and use these to construct the matrices $\mathbf{H}^<$ and $\mathbf{H}$ defined by

$$\mathbf{H}^< = \begin{pmatrix} s_1 & s_2 & s_3 & \ldots & s_k \\ s_2 & s_3 & s_4 & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ s_k & \ldots & & & s_{2k} \end{pmatrix} \tag{13}$$

and

$$\mathbf{H} = \begin{pmatrix} s_0 & s_1 & s_2 & \dots & s_{k-1} \\ s_1 & s_2 & s_3 & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ s_{k-1} & \dots & & & s_{2k-2} \end{pmatrix}. \tag{14}$$

It is possible to define a monic polynomial [16], $\phi_N(z)$ of degree N having the same roots as $f(z)$ if we set

$$\phi_N(z) = \prod_{j=1}^{N} (z - Z_j). \tag{15}$$

From the residue theorem [22] we have that

$$\langle z^p, \phi_N \rangle = 0, \quad p = 0, \dots, N-1. \tag{16}$$

Clearly, the goal is to derive a method, using the above properties, to see if we can obtain the polynomial in Eq. (15) and therefore the roots Z_k. In order to do this, it is more convenient to write Eq. (15) in the form

$$\phi_N(z) = \sum_{j=0}^{N} u_j z^j, \tag{17}$$

where the u_j are sums of products of the Z_k and where $u_N = 1$ because $\phi_N(z)$ is a monic polynomial.

From this representation, we may derive, using Eq. (16), a set of N linear equations for the unknowns u_j, $j = 0, \dots, N-1$.

A well-known theorem in linear algebra [24] shows that the roots of a monic polynomial are also the eigenvalues of its companion matrix. So, the roots of $f(z)$ are given by

$$\mathbf{C}_N \mathbf{x} = Z_k \mathbf{x}, \tag{18}$$

where $\mathbf{C}_N$ is defined as

$$\mathbf{C}_N = \begin{pmatrix} 0 & 0 & \dots\dots & 0 & -u_0 \\ 1 & 0 & \dots\dots & 0 & -u_1 \\ 0 & 1 & \dots\dots & 0 & -u_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots\dots & 1 & -u_{N-1} \end{pmatrix}. \tag{19}$$

This form is of no practical use because we do not know the u_j. However, multiplying Eq. (18) from the left by $\mathbf{H}_N$ yields

$$\left(\mathbf{H}_N^{<} - e\mathbf{H}_N\right)\mathbf{x} = 0, \tag{20}$$

where we have used the relationship that

$$\mathbf{H}_N^{<} = \mathbf{H}_N \mathbf{C}_N. \tag{21}$$

The proof of this result requires a rearrangement of the linear equations that are generated whenever Eq. (17) is substituted into Eq. (16).

Formally, the solutions of $f(z) = 0$ are given by the generalized eigenvalues e. Note that this theory takes care of repeated roots, that is duplicate values e will be possible. We have used the standard LaPack routine ZGEGV to perform the eigenvalue computation.

The integrals s_i are computed, in our code, in two ways. When the root counting is performed the function values are computed at the `npts` points along the contour and therefore can immediately be used in the trapezoidal rule. Thus root counting and integral evaluation can be performed efficiently in one step. We store the computed function values in a table and we have used FORTRAN DO loops to compute the integrals. This technique exploits both the cache memory and the pipelines in the Intel Pentium processor on which we have performed our tests. Note that in this situation, the choice of `npts` while also determining the accuracy of the computed integrals, could be matched precisely to the multiples of the cache size on the processor used to leverage maximum efficiency. Alternatively, and as a cross check on the trapezoidal integrals when required, we have implemented integral evaluations using the routine DQAG from QUADPACK; the user selects the integration scheme via the input.

3.3. Refining the roots with Halley's method

Generally, the iteration functions in Eq. (2) require the evaluation of derivatives with perhaps the most widely known such iteration technique being the Newton method [10]. Halley's iteration formula [11] for $f(z) = 0$ may be derived by applying Newton's method to the form

$$\frac{f(z)}{\sqrt{f'(z)}} \tag{22}$$

and may be shown to be cubically convergent [25] whereas Newton's method is not. Traub [10] has presented modified versions of several iteration functions for the situation in which derivatives are not available and we have employed Halley's method in its derivative free form.

We have used a table based approach in the routine HALLEY3 to compute and track the iterations of the Halley method. The use of a table eliminates, at the expense of a small amount of RAM, the need to copy variables at the end of each iteration in preparation for the next, and indeed also facilitates analysis and debugging of the process for complicated functions. For each iteration, z_m we compute and store the following values into a table:

$$z_m - z_{m-1}, \quad z_m - z_{m-2}, \quad f(z_m) \quad \text{and} \quad f(z_m) - f(z_{m-1}). \tag{23}$$

Clearly, the only significant computation here is $f(z_m)$ because $f(z_{m-1})$ is already available in our table from the previous iteration.

A new approximation for the root is given [10] by

$$z_{m+1} = z_m - \frac{f(z_m)}{D}, \tag{24}$$

where

$$D = f[z_m, z_{m-1}] - f(z_{m-1}) \times \frac{f[z_m, z_{m-1}, z_{m-2}]}{f[z_m, z_{m-1}]}, \tag{25}$$

where $f[z_m, z_{m-1}]$ and $f[z_m, z_{m-1}, z_{m-2}]$ are the finite difference analogues of $f'(z)$ and $f''(z)$, respectively, given by

$$f[z_m, z_{m-1}] = \frac{f(z_m) - f(z_{m-1})}{z_m - z_{m-1}} \tag{26}$$

and

$$f[z_m, z_{m-1}, z_{m-2}] = \frac{f[z_m, z_{m-1}] - f[z_{m-1}, z_{m-2}]}{(z_m - z_{m-2})}. \tag{27}$$

Production of the new iterate, z_{m+1} as defined in Eq. (24) only requires subtraction and division operations and thus is very rapid indeed because the values involved are already held in the processor's data cache.

Following Traub [10] we have also implemented the Aitken acceleration formula as follows:

$$z_{\text{Aitken}} = z_{m-2} - \frac{(z_{m-1} - z_{m-2})^2}{z_m - 2z_{m-1} + z_{m-2}}. \tag{28}$$

However, to generate and examine this result requires extra CPU cycles and an extra function evaluation $f(z_{\text{Aitken}})$, of course. Generally, we have found that it is just as efficient to execute the program without Aitken acceleration.

3.4. Points to note

There are two steps in the algorithm that can give rise to problems. Firstly, the sub-division of $\mathbb{W}$ into smaller rectangles is complicated by the fact that the newly introduced contours around the small sub-rectangles may run through a root. The code attempts to monitor this situation and makes a number of attempts to adjust the contours accordingly. The algorithm may however fail for situations where there is a dense mesh of roots in the middle of a rectangle and in such cases the code will terminate with a warning message. One way to circumvent this for clusters of a few roots is to choose the starting contour to be slightly larger or smaller.

Secondly, a word of caution is necessary regarding the Halley iterations. There is actually no guarantee that the isolated root estimates computed from the matrix diagonalization actually belong to the appropriate subset of the filled-in Julia set, K_c such that the Halley iterations may make matters worse. The finite difference implementation of Halley's method requires that we also choose two values slightly displaced from the estimated root. Either of these could also lie outside K_c. We comment on an example of this behavior for the Wilkinson polynomial in test case 2 below.

4. Implementation details

We have implemented the code in Fortran 90 in a traditional procedural fashion. and have not therefore used any of the object oriented facilities provided by the language We have made extensive use of the dynamic memory allocation facilities in the language to reduce to a minimum the number of fixed dimensions in the code and thereby making the code more adaptable to problem size and this is controlled by the user input. We have found this to be particularly useful when running multiple simultaneous instances of the program, for example, from a Perl script, because we can match the number of concurrent instances to the RAM in our computer thereby minimizing the paging overhead on the operating system.

Following common software engineering practice, the code performs rigorous cross checking, in so far as one can, to detect errors during various the stages of the computations. We have adopted a policy that if any error is found, the code prints a description and then terminates at that point. This is particularly relevant to the following situations:

- the contour sub-division is unsuccessful;
- the adaptive integrations cannot converge;
- the number of roots is over estimated;
- the Halley iterations cannot converge for any given root including the situation where the iterations diverge outside the contour.

4.1. User interface

The program first of all looks for a namelist DIELECTRIC at the start of the file attached to unit 5; this describes the arrangement in Fig. 1 Namelist DIELECTRIC should be followed by one or more instances of namelist INPUT, each of which specifies a region $\mathbb{W}$ within which roots are sought. The values specified in DIELECTRIC are used for all instances of INPUT. The code will loop over these instances until the end of file is detected. When testing the program with the *well-known* functions that we have provided, DIELECTRIC is not required. The program will then simply read the instances of INPUT in the file.

The program will write output to unit 6 and thus the code can be executed at the command line of a Unix shell as follows:

prog.x < input.dat > output.dat

4.1.1. Namelist DIELECTRIC

The following is a description of each variable in namelist DIELECTRIC. and its default value. It is only necessary to define, in the input file, those values that the user wishes to change from their defaults.

lambda The wavelength at which the relative permittivity is computed for the metal layer.
Type: double precision. Default: 0.0.

epsilon1 Relative permittivity of the upper layer in Fig. 1.
Type: double precision. Default: 0.0.

epsilon2 Relative permittivity of the lower layer in Fig. 1.
Type: double precision. Default: 0.0.

cfilename Name of the file holding the relative permittivity of the metal layer in Fig. 1. The format of each line is

wavelength Real part of ϵ_m Imaginary part of ϵ_m

Lines beginning with the ! character are ignored and therefore permit comments in the file.
Type: character*80. Default: blank.

inunit Fortran unit number to be used when opening the file defined in `cfilename`. Obviously this should never be 5 or 6.
Type: integer. Default: 70.

d The thickness of the metal layer.
Type: double precision. Default: 0.0.
a1 The thickness of the upper layer of dielectric.
Type: double precision. Default: 0.0.
a2 The thickness of the lower layer of dielectric.
Type: double precision. Default: 0.0.

The values of λ, d, a_0 and a_1 have the dimension of length and should always be specified in the same units. This is because these values arise in Eq. (8) as ratios (e.g., $k_0 d$).

4.1.2. Namelist INPUT

The following is a description of each variable in the input and its default value. It is only necessary to define, in the input file, those values that the user wishes to change from their defaults because the NAMELIST is initialized before being read each time.

title This string may be used to provide some descriptive information on the data. This string is printed on the output on unit 6 but is otherwise not processed.
Type: character*80. Default: blank.

ifunct The function, $f(z)$, for which the roots are being sought, must be programmed into the sub-routine FDF() in code. In order to provide extensible test capabilities with known functions, FDF() contains code for many functions. Therefore, the user must provide a flag at run-time to select which of the functions should be used in the execution. **ifunct** plays this role. It is an integer that correlates to the selection statements in FDF(). The following test functions are provided:
1 $f = \log(z) - z + 1.195281 + 0.536353\mathrm{i}$
2 $f = z^{-0.5z} - z + 2.195093245 - 2.750498700\mathrm{i}$
3 $f = 54 + z(44 + z(20 - z(3 - z)))$
4 $f = 15 + z(7 - z(16 + z(8 - z(1 + z))))$
5 $f = -36 + z^2(49 - z^2(14 - z^2))$
6 $f = 8 - z(6 - z(6 + z(1.0 + z(2 - z(2 - z)))))$
7 $f = \sin((\frac{1}{2} - \frac{1}{2}\mathrm{i})\pi z)$
8 $f = \sin(z)$
9 $f = \exp(3z) + 2z\cos(z) - 1$
When applying the code to the dispersion function described in a previous section, the user should set ifunct = 16. Type: integer. Default: 0 (this will cause the code to terminate with an error).

point This is a two-dimensional array defining the real and imaginary parts of the point at the lower left-hand side corner of the rectangle W corresponding to the domain $\mathbb{W}$. Element (1) is the real part, element (2) the imaginary.
Type: double precision. Default: 0.0, 0.0.

step This is a two-dimensional array defining the length of the rectangle along the real and imaginary axes. Element (1) is the real part, element (2) the imaginary.
Type: double precision. Default: 0.0, 0.0.

droot_converged This is the threshold for declaring a root. If $|f(z)|$ is found to be less than this value, then the code assumes that a root has been found.
Type double precision. Default: 10^{-10}.

zaitken When set to *.true.* the code will use the Aitken procedure at each iteration of the Halley method.
Default: *.false.*

zadaptive When set to *.true.*, the QUADPACK adaptive quadrature routine DQAG is used. When set to *.false.*, fixed point integration (trapezoidal rule) is used.
Default: *.false.*

npts The number of points per side of rectangle to be used in monitoring the change of argument around the contour and optionally in the trapezoidal integrations.
Type integer. Default: 0.

maxboxes The maximum number of sub-boxes into which $\mathbb{W}$ may be split.
Type integer. Default: 500.

max_roots_per_box The maximum number of roots allowed in a sub-box before attempting to compute the roots.
Type integer. Default: 5.

4.2. Code structure

The main program is a driver which reads the input namelist and then controls the computations. After the contour integrals have been computed, the main program builds the $\mathbf{H}^<$ and $\mathbf{H}$ matrices. As this is a simple data copying exercise, it is not necessary to make it into a subroutine. The following describes the relevant worker routines in the code.

COUNTZ evaluates the $f(z)$ around the contour in order to count the number of roots; thereby this routine also computes the integrals needed to build the matrices defined in Eqs. (13) and (14).

FDF computes $f(z)$ at the point z. It is used extensively from both the routines COUNTZ and HALLEY3.

ZGEGV this is the same routine used by Kravanja et al. [22] to solve the generalized eigenvalue problem. It is part of the LaPack library.

HALLEY3 applies Halley's iterative method as described above. In highly optimized environment, it may be necessary to make this an in-line function to avoid the penalties imposed by subroutine call overheads.

BOOL_IN_BOX is a simple logical function to return the value TRUE whenever a given point is found to lie outside the domain $\mathbb{W}$. This is used in the Halley iterations to ensure that we have not strayed outside the specified domain.

5. Test runs

We have executed test runs on a on Dell Optiplex GX270 PC which has a 3.2 GHz Intel Pentium processor, 1 Gbyte of RAM and 80 Gbytes of hard disk. The operating system, executing on the PC, was the Fedora Core 2 version of Linux. Specifically, this means that the 2.6.5 kernel was used. The code was compiled using version 8.1 of the Intel Fortran 90 for Linux compiler. We have used the following compiler options in all test runs:

-check bounds -warn all -free -w90 -e90

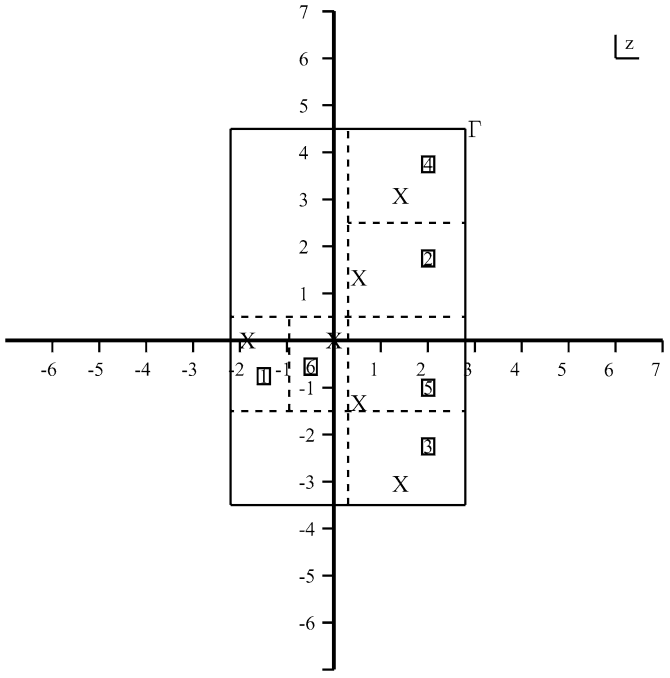

Fig. 3. Pictorial representation of the roots of the function $f = e^{3z} + 2z\cos z - 1$ reported in [22,26]. The rectangular contour surrounding the domain $\mathbb{W}$, used in our test input is shown as a bold line. The smaller rectangles produced by the sub-division algorithm in the code are shown bounded by dashed lines and numbered $1, \ldots, 6$ as reported on the program output.

Taken together, we believe that these provide a through check of the quality of the Fortran 90 implementation.

We provide two test cases with known results so that we may compare the performance of our program with those implementing other methods and we also provide one test application from our own work in electromagnetic field computation as described earlier in the paper.

5.1. Test case 1

For our first test case, we chose an example for which there are already published results [22,26]. This is the function

$$f = e^{3z} + 2z\cos z - 1 \tag{29}$$

and the location of the roots is illustrated by the Xs in Fig. 3. The following input data was used for this test:

```
&INPUT
 title='Example roots',
 ifunct = 9,
 point = -2.2d+00, -3.5d+00,
 step = 5.0d+00, 8.0d+00,
 droot_converged = 1.0d-10,
 zaitken =.false., zadaptive =.false.,
 npts = 8192, maxboxes = 500, max_roots_per_box
 = 1,
 &END
```

It is available in file `test_case_1.data`. In this test we request that $\mathbb{W}$ be broken down into smaller regions each of which has only one root. This is a severe restriction which is not strictly necessary as we have found that the algorithm is stable up to about five roots per sub-rectangle in general. On the other hand, this input tests the sub-division part of the algorithm. The sub-divisions produced by the code are shown in Fig. 3.

The following is an extract of the salient details from the full output of the program and highlights the computed results with the value of ABS($f(z)$) at these roots.

```
Solving for roots in box number: 1
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (-0.1844233953262D+01, -0.9219714845741D-14) 0.6396414D-13
Solving for roots in box number: 2
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (0.5308949302943D+00, 0.1331791876749D+01) 0.4517411D-10
Solving for roots in box number: 3
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (0.1414607177658D+01, -0.3047722062627D+01) 0.8526513D-13
Solving for roots in box number: 4
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (0.1414607177658D+01, 0.3047722062627D+01) 0.3552714D-13
Solving for roots in box number: 5
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (0.5308949302925D+00, -0.1331791876748D+01) 0.5963637D-10
Solving for roots in box number: 6
Final converged roots and ABS(f(z)), converged to 0.1000000D-09
1 (0.8360699803088D-13, 0.5978465798922D-13) 0.5139758D-12
End of file detected when reading &INPUT
**** Completed - Zeros Computation
```

These results are essentially identical to those reported by Kravanja et al. [22,26].

5.2. Test case 2: Modified Wilkinson polynomial

Finding the roots of the Wilkinson polynomial is a challenging test for any numerical method because of the large coefficients that arise if the polynomial is expanded. Bini [27], for example, has considered this situation using arbitrary precision numerics. So, to test our program, we have we created a modified version of the Wilkinson polynomial as follows

$$f(z) = \prod_{i=1}^{20}(z-i)(z-5)(z-6)^2(z-7)^3 \tag{30}$$

which has a double root at $z = 5$, a triple root at $z = 6$ and a quadruple root at $z = 7$. We also used this as a stress test of algorithm to see how many roots can be computed simultaneously; this means that we set the input parameter *max_roots_per_box* to a large value to prevent contour division taking place.

The input data for these tests may be found in file `test_case_2.data`. supplied with the program. The computed results are presented in Table 1 for the case where we used the fixed quadrature technique for the integrations.

For the case of a single double root at $z = 5$ in Table 1, it is can be seen that the computed solutions agree to within the convergence threshold of $|f(z)|$. We tried to apply our program to calculate all the roots at $z = 5, 6$ and 7 (i.e. nine roots) simultaneously, still using the fixed quadrature technique, without dividing the contour that is by defining a suitable large rectangular contour, but found that were unable to do this. The initial approximations from the eigenvalue problem were within the

Table 1
Root computations for the modified Wilkinson polynomial presented in Eq. (30). The Halley iterations were converged to 1.0×10^{-13} without Aitken acceleration

Contour	Integration method	Converged roots
point = (4.5, −0.5)	Fixed	$(5.000000000000, -0.45 \times 10^{-15})$
step = (1.0, 1.0)	npts=8192	$(5.000000000000, -0.45 \times 10^{-15})$
point = (5.5, −0.5)	Fixed	$(5.999999999409, 0.18 \times 10^{-8})$
step = (1.0, 1.0)		$(6.000000000348, 0.16 \times 10^{-8})$
		$(6.000000000608, 0.15 \times 10^{-8})$
point = (4.5, −0.5)	Fixed	$(5.000000000000, 0.14 \times 10^{-15})$
step = (2.0, 1.0)		$(5.000000000000, 0.64 \times 10^{-14})$
		$(5.999999998701, 0.12 \times 10^{-8})$
		$(6.000000001522, 0.80 \times 10^{-9})$
		$(6.000000001991, -0.26 \times 10^{-9})$
point = (6.5, −0.5)	Fixed	$(6.999999741099, 0.17 \times 10^{-6})$
step = (1.0, 1.0)		$(6.999999787949, -0.19 \times 10^{-6})$
		$(7.000000176328, -0.20 \times 10^{-6})$
		$(7.000000187719, 0.20 \times 10^{-6})$
point = (5.5, −0.5)	Fixed	$(5.999999998548, 0.10 \times 10^{-8})$
step = (2.0, 1.0)		$(6.000000000482, -0.17 \times 10^{-8})$
		$(6.000000001343, 0.10 \times 10^{-8})$
		$(6.999999788180, -0.19 \times 10^{-6})$
		$(6.999999892937, 0.26 \times 10^{-6})$
		$(7.000000190819, -0.19 \times 10^{-6})$
		$(7.000000299263, 0.11 \times 10^{-6})$

contour, although a poor approximation to the true roots, but the Halley iterations quickly diverged outside of the contour.

When we switched to using the QUADPACK adaptive integration scheme we encountered problems with the simultaneous computation of the roots at $z = 6, 7$ corresponding to the last entry in Table 1. When we set the absolute and relative error parameters in the call to DQAG() to be both $\geqslant 1.0 \times 10^{-7}$ we noted that the computed values of the integrals shown in Eq. (12) differed in typically the sixth significant figure from those computed using the fixed quadrature. This small perturbation led to slightly different initial estimates for the roots but significant in the sense that Halley method diverged outside of the box in these cases. Setting the relative error parameters to be $< 1.0 \times 10^{-7}$ caused DQAG to report a roundoff error when computing the integral in Eq. (12) with $p > 11$.

This example indicates the sensitivity of the algorithm to both the number of roots being solved simultaneously and to the function behavior. From the results of these tests we derived a default value of *max_roots_per_box* to be 5.

5.3. Gold film surrounded by glass substrate

As a final example we show the application of our package to the solution of the model problem shown in Fig. 1. We consider the situation where the metal is a gold film of thickness 50 nm surrounded by two layers of glass, each of thickness 50 nm too, and with the upper layer having $\epsilon_1 = 1.5$ and the lower layer $\epsilon_2 = 1.0$. The relative dielectric permittivity of gold as a function of wavelength in vacuum is provided as a data table in file `Au180.dat` with points ranging from 339.7 to 1240 nm wavelength. During the root finding procedure, the values of ϵ_m are computed by interpolation on the table values to the required wavelength.

We choose the incident wave to have a wavelength of 350.0 nm which yields, by interpolation, $\epsilon_m = (-0.388719, 6.055995)$. The search domain for γ is chosen, from the physics of the problem, to be

$$1.900 \leqslant \mathrm{Re}(\gamma) \leqslant 2.300, \tag{31}$$

$$-1.500 \leqslant \mathrm{Im}(\gamma) \leqslant -1.300. \tag{32}$$

Using 1024 points per side for root counting and integral computations and then converging the Halley iterations to 1.0×10^{-7}, the code locates a root at

$$\gamma = 2.0561045 - 1.4552085\mathrm{i}. \tag{33}$$

Performing the integrations using the adaptive scheme from QUADPACK produced the same result.

6. Concluding remarks

We have presented a new package, written in Fortran 90, which computes estimated solutions to the problem $f(z) = 0$ where $f(z)$ is defined over a finite rectangular domain $\mathbb{W}$ in the complex plane. Computation of $f'(z)$ is not performed but it is a fundamental requirement that $f(z)$ be analytic within and on the boundary of $\mathbb{W}$. Our package is targeted at computing problems where the evaluation of the derivative is computationally expensive. Our package computes estimated solutions and the derivative free version of Halley's method is used to iteratively refine the computed estimates to a higher degree of accuracy, should that have been specified by the user. We have used our program on a number of test functions and shown that our results represent the known roots. We have also presented an application of our program to eigenmode analysis for metal film optical waveguides.

The bulk of work in program revolves around the computation of $f(z)$ at small sized steps around the closed rectangular contour Γ surrounding $\mathbb{W}$. Ultimately, our package is sensitive only to the size of these steps. We have implemented a fixed quadrature method to compute the integrals because it has a deterministic number of compute steps but have also provided an adaptive quadrature scheme from QUADPACK as a crosscheck on the quality of the fixed scheme. For some complicated functions the adaptive quadrature scheme may be more efficient. The specific choice can however only be made after some experimentation.

As we have pointed out, one of the limiting factors on our method is the representation of floating point arithmetic used by the computer processor; this will of course always be a limitation in high performance codes and was also observed in the work of Botten et al. [15]. However, we are presently reimplementing our program to exploit arbitrary precision arithmetic. This will provide us with a tool which will help in the analysis of our method when it is applied to some computationally demanding functions. We are also working on a new

algorithm to handle the case, as outlined in the introduction, where there are roots lying very close to the contour.

Finally, we remind the user that if the derivatives of $f(z)$ can be computed in a timely fashion then it is probably better to use an alternative package which exploits that fact. Clearly, the more information that one has about the behavior of a function the easier the computational task of finding its roots; in our version of Halley's method, we estimate derivatives numerically. Nevertheless, there are several applications spaces in which our approach is the only viable method of solving $f(z) = 0$.

Acknowledgements

The Institute for Electronics, Communications and Information Technology (ECIT) is jointly funded by The Queen's University of Belfast, Invest Northern Ireland and The Department of Employment and Learning for Northern Ireland.

References

[1] D. Schmitt, R. Schuhmann, T. Weiland, Int. J. Numer. Model. Electron. Netw. Device Fields (UK) 8 (1995) 385–395.
[2] J. Arroyo, J. Zapata, IEEE Trans. on Microwave Theory and Techniques MTT-46 (1998) 1115–1123.
[3] R. Lehmensiek, P. Meyer, ACES J. 16 (2001) 1–10.
[4] I. Yamada, N.K. Bose, IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications 49 (2002) 298–304.
[5] S.-A. Teo, M.-S. Leong, S.-T. Chew, B.-L. Ooi, IEEE Trans. on Microwave Theory and Techniques MTT-50 (2002) 440–445.
[6] A.G. Schuchinsky, E. Brennan, V.F. Fusco, in: Fifth IEE International Conference on Computation in Electromagnetics—CEM 2004 (CP 505) Stratford-upon-Avon, UK, 19–22 April 2004, vol. 145, ISBN: 0 86341 400 1.
[7] R. Rodrígues-Berral, F. Mesa, F. Medina, Int. J. of RF and Computer-Aided Engineering 14 (2004) 73–83.
[8] C.J. Noble, M. Dörr, P.G. Burke, J. Phys. B: At. Mol. Opt. Phys. 26 (1993) 2983.
[9] C.J. Gillan, J. Tennyson, P.G. Burke, in: W.M. Huo, F.A. Gianturco (Eds.), Computational Methods for Electron-Molecule Collisions, Plenum, New York, ISBN 0-306-44911-0, 1994, pp. 239–254.
[10] J.F. Traub, Iterative Methods for the Solution of Equations, Prentice-Hall, Englewood Cliffs, NJ, ISBN 0-8284-0312-0, 1964. Republished by the American Mathematical Society 1982.
[11] E. Halley, Philos. Trans. Roy. Soc. London 18 (1694) 136.
[12] A. Douady, Julia sets and the Mandelbrot set, in: H.-O. Peitgen, D.H. Richter (Eds.), The Beauty of Fractals: Images of Complex Dynamical Systems, Springer-Verlag, Berlin, ISBN 0-387-15851-0, 1986, p. 161.
[13] C.A. Pickover, Comm. ACM 31 (1988) 1326–1329.
[14] L. Yau, A. Ben-Israel, Amer. Math. Monthly 105 (1998) 806.
[15] L.C. Botten, M.S. Craig, R.C. McPhedran, Comput. Phys. Comm. 29 (1983) 245.
[16] L.M. Delves, J.N. Lyness, Math. Comput. 21 (1967) 543.
[17] J. Herlocker, J. Ely, Reliable Comput. 1 (1995) 3.
[18] X. Yan, A.G. Schuchinsky, V.F. Fusco, in: IEEE High Frequency Postgraduate Student Colloquium, 5–6 Sept. 2005, pp. 35–38.
[19] M. Dellnitz, O. Schutze, Q. Zheng, J. Comput. Appl. Math. 138 (2002) 325.
[20] M.P. Carpentier, A.F. Dos Santos, J. Comput. Phys. 45 (1982) 210.
[21] P. Kravanja, M. van Barel, Computing 63 (1999) 69.
[22] P. Kravanja, M. Van Barel, O. Ragos, M.N. Vrahatis, F.A. Zafiropoulos, Comput. Phys. Comm. 124 (2000) 212.
[23] P. Kravanja, T. Sakurai, M. Van Barel, Computing 70 (2003) 335–347.
[24] J.E. Gentle, in: Numerical Linear Algebra for Applications in Statistics, in: Statistics and Computing, Springer, ISBN 0-387-98542-5, 1998, p. 123.
[25] J.B. Thoo, T.R. Scavo, Amer. Math. Monthly 102 (5) (1995) 416.
[26] P. Kravanja, PhD thesis, University Lueven, Belgium, 1999.
[27] D.A. Bini, Numer. Algorithms 13 (1996) 179–200.