Solving secular and polynomial equations: a multiprecision algorithm

Dario A. Bini, Dipartimento di Matematica Università di Pisa
`bini@dm.unipi.it`
Leonardo Robol, Scuola Normale Superiore, Pisa
`leonardo.robol@sns.it`

May 10, 2013

Abstract

We present an algorithm for the solution of polynomial equations and secular equations of the form $S(x) = 0$ for $S(x) = \sum_{i=1}^{n} \frac{a_i}{x-b_i} - 1 = 0$, which provides guaranteed approximation of the roots with any desired number of digits. It relies on the combination of two different strategies for dealing with the precision of the floating point computation: the strategy used in the package MPSolve of D. Bini and G. Fiorentino [Numer. Algo. 23, 2000] and the strategy used in the package Eigensolve of S. Fortune [J. Symb. Comput. 33, 2002]. The algorithm is based on the Ehrlich-Aberth (EA) iteration, and on several results introduced in the paper. In particular, we extend the concept and the properties of root-neighborhoods from polynomials to secular functions, provide perturbation results of the roots, obtain an effective stop condition for the EA iteration and guaranteed a posteriori error bounds. We provide an implementation, released in the package MPSolve 3.0, based on the GMP library. From the many numerical experiments it turns out that our code is generally much faster than MPSolve 2.0 and of the package Eigensolve. For certain polynomials, like the Mandelbrot or the partition polynomials the acceleration is dramatic. The algorithm exploits the parallel architecture of the computing platform.

Keywords: Secular equations, polynomial roots, multiprecision computations, Ehrlich-Aberth iteration, root neighborhoods

1 Introduction

Given a positive integer n and complex numbers a_i, b_i, $i = 1, \ldots, n$ such that $b_i \neq b_j$ for $i \neq j$ and $a_i \neq 0$ for $i = 1, \ldots, n$, consider the rational function

$$S(x) = \sum_{i=1}^{n} \frac{a_i}{x - b_i} - 1. \tag{1}$$

We associate with $S(x)$ the *secular equation* $S(x) = 0$. The numbers b_i and a_i, for $i = 1, \dots, n$ are referred to as the nodes and the coefficients of the secular function $S(x)$, respectively.

Equations of this kind are mainly encountered in the case of real nodes b_i and positive coefficients a_i when the roots are real. Typical examples are modifying symmetric eigenvalue problems [17], or solving the tridiagonal symmetric eigenvalue problem by means of divide and conquer techniques [11, 19], updating the singular values of a matrix, solving least squares problems [23], invariant subspace computation [16] and more. Over the complex field, for any set of nodes and coefficients, secular equations are encountered in the solution of the eigenvalue problem for a diagonal plus rank-one matrix [2, 6, 7, 8] and in representing generalized companion matrix pencils in the Lagrange basis especially in the framework of "polynomial algebra by values" [3].

Secular equations are also a powerful tool for attacking the polynomial root-finding problem. This is the main fact that motivates our interest in such equations. In fact, the monic polynomial of degree n

$$p(x) = \Pi(x)S(x), \quad \Pi(x) = -\prod_{i=1}^{n}(x - b_i)$$

has roots that coincide with the roots of $S(x)$. Moreover, one can verify that

$$a_i = \frac{p(b_i)}{\prod_{j=1,\, j\neq i}^{n}(x - b_j)} \tag{2}$$

so that, given the polynomial $p(x)$ and the nodes b_i it is not expensive to compute the coefficients a_i and to reformulate the polynomial root-finding problem in terms of a secular equation.

In this paper we present a method for the numerical solution of secular equations together with its computational analysis. More precisely we describe, analyze and implement an algorithm which, given in input the coefficients a_i and the nodes b_i, $i = 1, \dots, n$ of the secular function $S(x)$ together with an integer d, provides in output the roots of $S(x)$ represented with d guaranteed digits. In its cheaper version (isolation), the algorithm can provide approximations to the roots with the minimum number of digits sufficient to separate them from each other. The maximum number d of digits is used only for those roots, if any, which cannot be otherwise separated.

The algorithm can be effectively used as a tool for computing an arbitrarily large number d of digits of the roots of a polynomial $p(x)$ assigned either in terms of its coefficients in some polynomial basis or by means of a black box which, given in input a complex number x, provides in output the complex number $p(x)$. In fact we will show that using the representation of a polynomial in terms of secular equation provides substantial computational advantages.

The method relies on the combination of two different strategies to reduce the required precision of the floating point arithmetic: the strategy adopted in the package of MPSolve of D. Bini and G. Fiorentino [5], and that used in the package Eigensolve of S. Fortune [15]. It exploits some theoretical results, that we present in this paper, concerning root-neighborhoods, numerical conditioning, *a posteriori* error bounds, and rounding error analysis, related to computations with secular functions. It relies also on the formulation of the problem given in terms of structured matrices and on the Ehrlich-Aberth iteration as main approximation engine [1, 13].

The algorithm that we have obtained has been implemented in the language C and incorporated in the package MPSolve originating the release MPSolve 3.0. The software is free and can be downloaded from `http://riccati.dm.unipi.it/mpsolve`. It enables to deal with secular and polynomial equations where the real or complex input data can take either the approximate form of floating point numbers or the exact form of integers and rationals. The implementation exploits the parallel architecture of the computing platform.

From the many numerical experiments that we have performed our code, even though applied without the parallelism, is generally faster than MPSolve and Eigensolve. For certain polynomials it is dramatically faster. The speed up that it reaches when using multicore hardware is close to optimal. Just to make an example, for the partition polynomial of degree 72.000 which has integer coefficients representable with several megabytes, MPSolve 2.0 took about 30 days to compute all the roots [9]. Our code computes the roots in less than 2 hours whereas Eigensolve has an estimated CPU time of many years.

The paper is organized as follows. In Section 2 we recall the strategies of MPSolve and Eigensolve, and give an overview of our algorithm. In Section 3 we develop the numerical tools that we need. In particular we provide a backward stable method for computing $S(x)$, and we extend the definition and the properties of root-neighborhoods [4, 24] from polynomials to secular functions. These properties allow us to devise effective stop conditions to halt the Ehrlich-Aberth iteration. Section 4 deals with the matrix representation of the problem and with the way of constructing different secular functions having the same roots as $S(x)$ and using different sets of nodes. We refer to these functions as *equivalent functions*. Gerschgorin-like inclusion results are given and used for devising *a posteriori* error bounds. In Section 5 we perform a perturbation analysis of the roots of secular functions where we show that the condition number of the roots converges to zero as the nodes, used for representing the equivalent secular function, converge to the roots. The Ehrlich Aberth iteration is recalled in Section 6. Finally, in Section 7 we present the results of the numerical experiments.

2 Overview of the algorithm

Our algorithm performs computations in floating point arithmetic with a variable number of digits. Since high precision arithmetic is expensive, our goal is to keep the number of digits of the floating point computation as low as possible. We will refer to the *working precision* as to the number w of binary digits used in the current floating point computation and denote $u = 2^{-w}$ the corresponding *machine precision.* Recall that the standard IEEE double precision arithmetic has $w = 53$ bits.

Here we provide an overview of our algorithm. First, we recall two different strategies to manage the working precision for computing roots of polynomials: the strategy of MPSolve and the one of Eigensolve. Then we describe the new approach which combines the two different techniques. To better explain this, we need to anticipate the following tools and concepts which will be better clarified and investigated in the next sections.

Set of inclusion disks is a set of disks in the complex plane, say provided by the Gerschgorin theorem, such that their union contains all the roots, and any connected component formed by k disks contains k roots. This way, isolated disks contain only one root.

Newton-isolated disks. A disk in a set of inclusion disks is said Newton isolated, or simply *isolated*, if if its distance from the closest disk in the set is at least 3n times its radius. This way, if the disks include the roots of a polynomial $p(x)$, Newton's iteration applied to $p(x)$ starting from the center of a Newton isolated disk converges quadratically right from the start to the root in this disk in view of [26].

Ehrlich-Aberth (EA) iteration. It is defined by the sequence of vectors $x^{(k)} = (x_i^{(k)}) \in \mathbb{C}^n$ such that

$$x_i^{(k+1)} = x_i^{(k)} - \frac{N(x_i^{(k)})}{1 - N(x_i^{(k)}) \sum_{j=1,\, j\neq i}^{n} \frac{1}{x_i^{(k)} - x_j^{(k)}}}, \quad N(x) = p(x)/p'(x). \tag{3}$$

An analogous sequence can be generated in the Gauss-Seidel style. The Newton correction $N(x)$ can be expressed in terms of $S(x)$ as

$$N(x) = \frac{S(x)}{S(x) \sum_{i=1}^{n} \frac{1}{x - b_i} + S'(x)}. \tag{4}$$

If convergent, the sequence $x^{(k)}$ converges to the n-tuple of the roots of $p(x)$; convergence to simple roots is locally cubic, it is linear for multiple roots. There is no proof of the global convergence of the EA iteration, however, from the practical point of view, no results where the sequence fails to converge have been encountered. In the practical use of this iteration, some control can be set for non-convergence. A nice feature of the iteration (3) is that it can be applied selectively only for the subscripts i of interest. The cost per iteration is $O(nk)$ arithmetic operations (ops) where k is the number of subscripts i to which the iteration is applied. Another nice feature is that the iteration can be easily parallelized. The EA iteration performs implicit deflation of the roots without computing quotient and remainder and, unlike the iterations based on explicit deflation, it is self correcting.

Root-neighborhood. For a polynomial $p(x) = \sum_{i=0}^{n} p_i x^i$ and a given $\epsilon > 0$, the ϵ–root-neighborhood of $p(x)$ is the set formed by the roots of all the polynomials $\tilde{p}(x) = \sum_{i=0}^{n} \tilde{p}_i x^i$, where $\tilde{p}_i = p_i(1 + \epsilon_i)$, $|\epsilon_i| \leq \epsilon$. The ϵ–root-neighborhood of $S(x) = \sum_{i=1}^{n} \frac{a_i}{x - b_i} - 1$ is the set formed by the roots of the secular function $S(x) = \sum_{i=1}^{n} \frac{\tilde{a}_i}{x - b_i} - 1$, where $\tilde{a}_i = a_i(1 + \epsilon_i)$, $|\epsilon_i| \leq \epsilon$.

The general lines of the strategy on which MPSolve is based are reported below. Here, the goal is to arrive at isolating all the roots up to d guaranteed correct digits.

1. The EA iteration is applied with a working precision of $w = 53$ bits until all the approximations are in the ϵ-root-neighborhood for $\epsilon = \gamma u n$, $u = 2^{-w}$, and γ is a suitable constant whose value comes from the rounding error analysis of the Horner rule.

2. A set of inclusion disks is computed. If all the disks are isolated, then the algorithm stops and the approximations are delivered. The algorithm stops also if the overlapping disks, if any, have radii which guarantee at least d correct digits in the approximation.

3. If there are some overlapping or non-isolated disks (unsolved clusters), then the number of digits of the working precision is doubled, i.e., we set $w := 2w$, and the Ehrlich-Aberth iteration (3) is applied with the new higher precision *only* for the indices i corresponding to the approximations in these clusters until the computed approximations are in the ϵ–root-neighborhood, $\epsilon = \gamma u n$. The computation is continued from step 2

The algorithm delivers a set of inclusion disks where each disk is either isolated or its center provides a guaranteed approximation to a root with d digits. With this strategy, the working precision is tuned according to the closeness and the conditioning of the roots. High precision is used only where it is needed to zoom in and to solve the tight clusters if any. A polynomial with few clustered roots is not a difficult polynomial. In fact almost all the roots are computed with the standard floating point arithmetic and the more expensive large precision is used only for the few clustered roots. This approach can encounter slowdown in the case of polynomials having almost all ill-conditioned roots.

A different strategy is used by Eigensolve where the roots of $S(x)$, or equivalently of the polynomial $p(x)$, are viewed as the eigenvalues of the matrix $A(a, b) = \mathrm{diag}(b_1, \ldots, b_n) - ae^T$, for $a = (a_1, \ldots, a_n)^T$, $e = (1, \ldots, 1)^T$, and a_i defined in (2). This strategy for computing all the roots of $p(x)$ with d correct digits is outlined below.

1. The QR iteration is applied to the companion matrix associated with $p(x)$ with working precision $w = d$ until some approximations $\xi_1, \ldots, \xi_n$ to the roots are delivered.

2. (Regeneration) Set $b_i = \xi_i$ for $i = 1, \ldots, n$ and compute a_i by means of (2) with a sufficiently high working precision which guarantees at least w correct digits in the computed a_i.

3. Apply the QR iteration to the matrix $A(a, b)$ with working precision w until some new approximations of the eigenvalues of A are delivered. If a suitable stop condition is satisfied then output the approximations and exit. Otherwise continue from step 2.

The idea at the basis of this strategy is that the roots of $S(x)$ are better conditioned if the nodes b_i are close to the roots. The computation at step 2 must be performed in a higher working precision in order to keep under control the loss of accuracy in the computation of $p(x)$. The advantage of this approach is the fact that high precision is not required in the iterative part of the algorithm which is expected to be the most cumbersome. High precision is left only for the regeneration stage. This is particularly appreciated in the case of polynomials with many ill conditioned roots where MPSolve requires a large working precision in all the iterations.

However, there are some drawbacks of the Eigensolve approach. The first is that the QR iteration requires $O(n^3)$ arithmetic operations. The second is that this strategy cannot take full advantage from the existence of few ill conditioned roots. In fact, the existence of ill conditioned roots, independently of their number, implies a large number of regenerations in the above scheme. Another drawback is that the cheaper goal of isolating the roots cannot be fulfilled by this approach.

The algorithm that we propose combines the two above strategies and maintains the advantages of both MPSolve and Eigensolve. The new strategy is outlined in the algorithm below.

Algorithm 1.

1. Set $w = 53$, apply the EA iteration to $p(x)$ and obtain some preliminary approximations $\xi_1, \dots, \xi_n$ to the roots.

2. Set $b_i = \xi_i$, $i = 1, \dots, n$ and compute a_i from (2) with a sufficiently high precision which guarantees w correct digits in the computed a_i. The disks of center b_i and radius $n|a_i|$ provide a set of inclusion disks

3. If all the disks are Newton isolated or the overlapping disks have radii which guarantee at least w correct digits in the approximation proceed at step 4. Otherwise, with the working precision w and by relying on (4) with input data b_i and a_i, apply the EA iteration (3) to the polynomial $\Pi(x)S(x)$ *only* for those approximations which are not yet isolated. Halt the iterations when these approximations are in the ϵ–root-neighborhood for $\epsilon = \gamma u \log_2 n$, $u = 2^{-w}$. The constant γ is determined by the backward error analysis in the computation of $S(x)$. Continue from step 2.

4. Halt the iteration if all the disks are isolated or the overlapping disks have radii which guarantee at least d correct digits in the approximation. Otherwise double the working precision, i.e., set $w := 2w$ and continue from step 2.

With this strategy, we can preserve the advantages of the MPSolve approach since the higher precision is used only for the clustered roots by means of a selective application of the EA iteration and by using different levels of working precision. At the same time the algorithm keeps the advantages of Eigensolve by providing refined representations of the secular equation where the roots improve their conditioning as long as the nodes get close to the roots. Another advantage is that the cost per step is $O(nk)$ ops, where k is the number of the current unsolved approximations. Finally, the radius of the root-neighborhood is $O(u \log_2 n)$ whereas for the polynomial case (MPSolve) is $O(nu)$.

In order to implement this strategy we need to develop suitable tools and theoretical results that we describe below.

2.1 Main theoretical results

In the design of our algorithm we need some theoretical properties that provide effective tools for the actual implementation of the above strategy. Here we give an outline of the main theoretical results proved in the paper.

A rounding error analysis of the computation of $S(x)$ enables us to prove that the computed value $\mathrm{fl}_u\,(Sx)$ still contains some correct digits if

$$|\mathrm{fl}_u\,(Sx))\,| > u(1+\kappa_n)\sigma(x), \text{ for } \kappa_n = \lceil \log_2 n \rceil + 7\sqrt{2},\ \sigma(x) = \sum_{i=1}^{n} \left| \frac{a_i}{x-b_i} \right|.$$

Here, $\mathrm{fl}_u\,(\cdot)$ denotes the result obtained in floating point arithmetic with machine precision u. Therefore, if the above inequality is not satisfied (stop condition) then the iteration is halted since no improvement is expected.

The concept of ϵ–root-neighborhood introduced in [24] for polynomials and exploited in [4], is extended to the case of secular functions. In particular we prove that the ϵ–root-neighborhood of $S(x)$ coincides with the set $\{x \in \mathbb{C} : |S(x)| \le \epsilon\sigma(x)\}$. This fact allows us to prove that any approximation satisfying our stop condition belongs to the ϵ–root-neighborhood where $\epsilon = 2u\kappa_n\sigma(x)$. That is, the approximation is the exact root of a slightly perturbed secular equation.

In the case of polynomials, a similar result has been proved in [4] where the term κ_n is replaced by $(2\sqrt{2}+1)n+1$. The reduction from n to $\log_2 n$ shows that the secular equation approach is computationally more favorable.

Another theoretical result proved in the paper, fundamental for the effectiveness of our algorithm, shows that the condition number of the roots of $S(x)$, as functions of the coefficients a_i, converges to zero as the nodes b_i converge to the roots of $S(x)$. This property holds also for a multiple root if the convergence of the different approximation to the same multiple root respects some mild conditions. In our algorithm these conditions are satisfied by the way the approximations generated by the EA iteration converges to multiple roots [1]. This fact explains why approximations to multiple roots are not affected by the large condition number, and confirms the effectiveness of using secular functions instead of polynomials.

The algorithm relies on guaranteed *a posteriori* error bounds obtained by representing the roots in terms of the eigenvalues of a suitable structured matrix to which the Gerschgorin theorem is successfully applied.

As already mentioned, the main engine for approximating the roots is the EA iteration [1, 13] which is applied to the monic polynomial $p(x) = S(x)\Pi(x)$.

In principle, we might have applied the technology of semiseparable matrices [27] to implement the QR iteration at low cost. However, we did not make it since the available algorithms do not yet allow to deal with complex matrices in an effective, reliable and robust way.

3 Numerical issues

In this section we deal with two important facts about secular equations which are related to each other. The first is that the evaluation of a secular equation in floating point arithmetic with machine precision u can be performed by means of backward stable algorithms with backward errors which grow as $u \log_2 n$. The second concerns the analysis of the ϵ–root-neighborhood of a secular function $S(x)$. As a consequence, we provide an effective stop condition for any iteration which approximates the secular roots relying on the value $\widetilde{S}(x)$ actually computed in floating point arithmetic in place of $S(x)$.

Instead of performing a first order error analysis, where the error bounds are reported only in their linear part in u, we provide exact bounds which hold for any value of the machine precision u. For this reason we introduce the useful function $\nu(k) = k/(1-ku)$. The reader interested in a first order error analysis may simply replace $\nu(k)$ with k. We also use the symbol $\doteq$ to denote equality of the linear parts of the error bounds so that we may write $\nu(k)u \doteq ku$. Similarly we do with $\dot{\leq}$. We refer the reader to Sections 3.4 and 3.6 of the book [20] for more details. We recall the following result from [20] which resumes the basic properties that we need for our rounding error analysis.

Proposition 2. *For any positive k such that $ku < 1$ define $\nu(k) = \frac{k}{1-ku}$ and let ν_k denote a quantity bounded according to $|\nu_k| \leq \nu(k)u < 1$. Then*

$$\begin{aligned} &(1+\nu_k)(1+\nu_j) = 1 + \nu_{k+j} \\ &\tfrac{1+\nu_k}{1+\nu_j} = 1 + \nu_{k+j}, \end{aligned} \tag{5}$$

as long as $0 < (k+j)u < 1$. Moreover, for the floating point operations between complex numbers it holds that

$$\begin{aligned} &\mathrm{fl}_u\,(a+b) = (a+b)(1+\epsilon_{\pm}), \\ &\mathrm{fl}_u\,(ab) = ab(1+\epsilon_*), \quad \mathrm{fl}_u\,(a \div b) = a \div b(1+\epsilon_{\div}), \\ &|\epsilon_{\pm}| \leq u, \quad |\epsilon_*| \leq \nu(2)\sqrt{2}u, \quad |\epsilon_{\div}| \leq \nu(7)\sqrt{2}u. \end{aligned} \tag{6}$$

Proof. Concerning equation (5) we provide a proof which holds also for non-integer values of k as otherwise required in [20]. Define $f(x) = \frac{x}{1-x}$ so that $\nu(k)u = f(ku)$. Then, to show that $(1+\nu_k)(1+\nu_j) = 1+\nu_{k+j}$, it is enough to prove that for any pair $a, b > 0$ of reals such that $a + b < 1$, it holds that $(1 + f(a))(1 + f(b)) \leq 1 + f(a + b)$. Since $1 + f(a) = \frac{1}{1-a}$, the latter inequality turns into $(1-a)(1-b) \geq 1-a-b$ which is trivially satisfied. Similarly, the equation $\frac{1+\nu_k}{1+\nu_j} = 1+\nu_{k+j}$ is equivalent to $\frac{1+f(a)}{1+f(b)} \leq 1+f(a+b)$ which is satisfied since $(1-b)(1-a-b) \leq 1-a$ for any $a, b \geq 0$ such that $0 \leq a+b < 1$. □

Given a vector $t = (t_i) \in \mathbb{C}^n$ consider the following algorithm for computing $\mathrm{sum}(t) = \sum_{i=1}^n t_i$:

$$\begin{aligned} &\mathrm{sum}(t) = \mathrm{sum}(t_+) + \mathrm{sum}(t_-), && \text{if } n > 1, \\ &\mathrm{sum}(t) = t, && \text{if } n = 1, \\ &t_+ = (t_1, \ldots, t_m),\ t_- = (t_{m+1}, \ldots, t_n),\ m = \lceil n/2 \rceil. \end{aligned} \tag{7}$$

Denote $\mathrm{fl}_u(\mathrm{sum}(t))$ the value obtained by performing the algorithm in floating point arithmetic with precision u.

By applying Proposition 2 it is easy to see that the computation (7) is backward stable that is

$$\mathrm{fl}_u(\mathrm{sum}(t)) = \textstyle\sum_{i=1}^n t_i(1+\delta_i), \quad |\delta_i| \le u\nu(\lceil \log_2 n \rceil). \tag{8}$$

The value $S(x)$ of the secular function S at x can be computed in the following way

$$\begin{aligned} &t_i = a_i/(x - b_i), \quad \text{for } i = 1, \ldots, n, \\ &S(x) = \mathrm{sum}(t) - 1. \end{aligned} \tag{9}$$

This algorithm performs the computation in $2n$ additions and n divisions. By applying (6), it turns out that the actual value $\mathrm{fl}_u(t)$ computed in place of t at each step of this algorithm, when using floating point arithmetic, is given by

$$\mathrm{fl}_u(t) = \frac{a_i}{x - b_i} \frac{1 + \epsilon_{\div}^{(i)}}{1 + \epsilon_{\pm}^{(i)}}$$

where $\epsilon_{\div}^{(i)}$ and $\epsilon_{\pm}^{(i)}$ are the local errors of division and subtraction. That is, in view of Proposition 2, one has

$$\mathrm{fl}_u(t) = \frac{a_i}{x - b_i}(1 + \epsilon_i), \text{ where } |\epsilon_i| \le \nu(1 + 7\sqrt{2})u \doteq (1 + 7\sqrt{2})u.$$

Summing up all the terms and using (8) yields the following expression

$$\mathrm{fl}_u(S(x)) = (\sum_{i=1}^n \frac{a_i(1+\delta_i)(1+\epsilon_i)}{x - b_i} - 1)(1 + \delta) \tag{10}$$

where δ is the local error generated in computing the last subtraction such that $|\delta| \le u$. By applying (5) to equation (10) we obtain the following result concerning the backward stability of Algorithm (9).

Proposition 3. *Algorithm* (9) *for the evaluation of the secular function*

$$S(x) = \sum_{i=1}^n \frac{a_i}{x - b_i} - c$$

is such that

$$\mathrm{fl}_u\,(S(x)) = \left(\sum_{i=1}^{n} \frac{a_i}{x-b_i}(1+\theta_i) - c\right)(1+\delta)$$

where $|\delta| \le u$, $|\theta_i| \le \kappa_n u$, $\kappa_n = \nu(\lceil \log_2 n\rceil + 7\sqrt{2})$. *That is,* $\mathrm{fl}_u\,(S(x)) = \sum_{i=1}^{n} \frac{\tilde a_i}{x-b_i} - \tilde c =: \widetilde{S}(x)$, *where* $\tilde a_i = a_i(1+\delta_i)(1+\delta)$, $\tilde c = c(1+\delta)$.

From the above result we obtain that

$$\mathrm{fl}_u\,(S(x)) = (1+\delta)(S(x) + \sum_{i=1}^{n} \frac{a_i}{x-b_i}\theta_i). \tag{11}$$

This equation provides the useful properties reported in the following

Corollary 4. *For the values* $S(x)$ *and* $\mathrm{fl}_u\,(S((x))$ *the following inequalities hold*

$$\begin{aligned} |S(x)| &\le \frac{1}{1-u}\,|\mathrm{fl}_u\,(S(x))| + u\kappa_n\sigma(x), \\ |\mathrm{fl}_u\,(S(x))| &\le (1+u)\,|S(x)| + u(1+u)\kappa_n\sigma(x). \end{aligned} \tag{12}$$

Moreover

$$\begin{aligned} \frac{\mathrm{fl}_u\,(S(x)) - S(x)}{\mathrm{fl}_u\,(S(x))} &= \delta\frac{S(x)}{\mathrm{fl}_u\,(S(x))} + (1+\delta)\frac{1}{\mathrm{fl}_u\,(S(x))}\sum_{i=1}^{n}\frac{a_i}{x-b_i}\theta_i, \\ \left|\frac{\mathrm{fl}_u\,(S(x)) - S(x)}{\mathrm{fl}_u\,(S(x))}\right| &\le u(\nu(1) + \frac{\kappa_n\sigma(x)}{\mathrm{fl}_u\,(S(x))}) + 2u^2\frac{\kappa_n\sigma(x)}{\mathrm{fl}_u\,(S(x))}. \end{aligned} \tag{13}$$

Proof. The first two inequalities (12) as well as the equation in (13) are immediate consequences of (11). To prove the inequality in (13) take the moduli of both sides of (13), apply the triangle inequality and obtain that

$$\left|\frac{\mathrm{fl}_u\,(S(x)) - S(x)}{\mathrm{fl}_u\,(S(x))}\right| \le u\left|\frac{S(x)}{\mathrm{fl}_u\,(S(x))}\right| + \frac{1}{\mathrm{fl}_u\,(S(x))}(1+u)k_n u\sigma(x).$$

Deduce from (12) that

$$\left|\frac{S(x)}{\mathrm{fl}_u\,(S(x))}\right| \le \frac{1}{1-u} + u\left|\frac{k_n\sigma(x)}{\mathrm{fl}_u\,(S(x))}\right|.$$

Replace the latter inequality in the former and obtain the result. □

Remark 5. The inequalities provided in the above corollary, are strict in the sense that they turn to equalities for specific choices of the values δ and θ_i satisfying the conditions $|\delta| = u$, $|\theta_i| = \kappa_n u$.

3.1 Stop condition

An important issue encountered in the implementation of numerical root-finders based on iterative processes is when to halt the iteration. In general, if the computed value $\mathrm{fl}_u(S(x))$ still contains information, it is worth continuing the iteration. This happens if the relative error $\left|\frac{S(x)-\mathrm{fl}_u(S(x))}{\mathrm{fl}_u(S(x))}\right|$ of the computation is less than 1. In fact, in this case at least one bit of information is contained in the computed value $\mathrm{fl}_u(S(x))$.

Corollary 4 provides a means to implement a stop condition based on the relative error estimate in the computation of $\mathrm{fl}_u(S(x))$. In fact, observe that, if

$$|\mathrm{fl}_u(S(x))| \le \kappa_n u\sigma(x)\frac{(1+2u)(1-u)}{1-2u} \doteq \kappa_n u\sigma(x), \tag{14}$$

then the upper bound to the modulus of the relative error provided in Corollary 4 is greater than or equal to one. In this case, there is no guarantee that the computed value $\mathrm{fl}_u(S(x))$ contains correct information. This way, condition (14) can be used as a test for halting the iterations in any secular root-finder which relies on the information contained in the computed value $\mathrm{fl}_u(S(x))$.

3.2 Root neighborhoods

Here we analyze the concept of ϵ root-neighborhood of $S(x)$, relate it to the properties of backward stability introduced in Proposition 3 and to the halt condition (14). Then we extend to secular equations the properties of root-neighborhoods proved in [4] in the case of polynomials.

Definition 6. *Let ϵ be a positive real number. Given the secular function $S(x)$ defined in* (1)*, we call* ϵ–secular-neighborhood *of $S(x)$ the set*

$$\mathrm{SN}_\epsilon(S) = \{\widehat{S}(x) = \sum_{i=1}^{n} \frac{\hat{a}_i}{x-b_i} - 1, \quad \hat{a}_i = a_i(1+\epsilon_i), \ |\epsilon_i| \le \epsilon\}.$$

We call ϵ–root-neighborhood *of $S(x)$ the set*

$$\mathrm{RN}_\epsilon(S) = \left\{\xi \in \mathbb{C} \mid \exists\, \widehat{S}(x) \in \mathrm{SN}_\epsilon(x) \textit{ such that } \widehat{S}(\xi) = 0\right\}.$$

The concept of root neighborhood replaces the concept of root in the case of secular equations where the coefficients a_i are approximately known within a relative error bounded by ϵ. In designing a floating point root-finder, the goal is to arrive at computing approximations to the roots of $S(x)$ belonging to $\mathrm{RN}_\epsilon(S)$ for some value of $\epsilon \ge u$ as close as possible to u. We will prove here that ϵ has the order of magnitude of $u\log_2 n$.

Recall that the roots of a polynomial are continuous functions of its coefficients. Therefore, since $p(x) = S(x)\Pi(x)$ has the same roots as $S(x)$, we deduce that the roots of $S(x)$ are continuous functions of the coefficients a_i. This fact, together with the above definition, allows us to prove the following

Proposition 7. *Let $S(x)$ be the secular function of* (1)*. The sets* $\mathrm{SN}_\epsilon(S)$ *and* $\mathrm{RN}_\epsilon(S)$ *satisfy the following properties*

1. $\mathrm{SN}_\delta(S) \subseteq \mathrm{SN}_\epsilon(S)$ *for* $0 < \delta \le \epsilon$*;*
2. $\mathrm{RN}_\delta(S) \subseteq \mathrm{RN}_\epsilon(S)$ *for* $0 < \delta \le \epsilon$*;*
3. $\mathrm{SN}_\epsilon(S)$ *is convex;*
4. *if* U_ϵ *is a connected component of* $\mathrm{RN}_\epsilon(S)$ *then all the secular function* $\widehat{S}(x) \in \mathrm{SN}_\epsilon(S)$ *have the same number of roots in* U_ϵ*;*
5. *if* $\mathrm{RN}_\delta(S)$ *is formed by* n *connected components then any secular function* $\widehat{S}(x) \in \mathrm{SN}_\epsilon(S)$ *has exactly one root in each component.*

Proof. Part 1 follows directly from the definition. Part 2 is an immediate consequence of part 1. If $\widehat{S}(x), \widetilde{S}(x) \in \mathrm{SN}_\epsilon(S)$ then $\hat{a}_i = a_i(1+\hat{\epsilon}_i)$, $\tilde{a}_i = a_i(1+\tilde{\epsilon}_i)$, with $|\hat{\epsilon}_i| \le \epsilon$ and $|\tilde{\epsilon}_i| \le \epsilon$, where $\hat{a}_i$ and $\tilde{a}_i$ are the coefficients of $\widehat{S}(x)$ and $\widetilde{S}(x)$, respectively. Therefore, the function $S_t(x) = t\widehat{S}(x) + (1-t)\widetilde{S}(x)$, for $0 \le t \le 1$ has coefficients $ta_i(1+\hat{\epsilon}_i)+(1-t)a_i(1+\tilde{\epsilon}_i) = a_i(1+\check{\epsilon}_i)$ where $\check{\epsilon}_i = t\hat{\epsilon}_i + (1-t)\tilde{\epsilon}_i$. Whence $|\check{\epsilon}_i| \le t\epsilon + (1-t)\epsilon = \epsilon$, that is $S_t(x) \in \mathrm{SN}_\epsilon(S)$. This completes the proof of part 3. Concerning part 4, let $\widehat{S}(x), \widetilde{S}(x) \in \mathrm{SN}_\epsilon(S)$ and define $S_t(x)$ as before. For the continuity of the roots of $S_t(x)$ as functions of t, it follows that the number of roots of $S_t(x)$ in the connected component U_ϵ is constant with respect to t. Otherwise some root would necessarily move out of U_ϵ and therefore out of $\mathrm{RN}_\epsilon(S)$. Hence $\widehat{S}(x)$ and $\widetilde{S}(x)$ must have the same number of roots in U_ϵ. Finally, part 5 follows since the connected components cannot be empty and there cannot be more than n roots of $S(x)$. □

For the sake of notational simplicity, in the following we write RN_ϵ in place of $\mathrm{RN}_\epsilon(S)$. Apparently, checking the condition $\xi \in \mathrm{RN}_\epsilon$ does not seem to be an easy task. In the next proposition we provide a computationally cheap way to test if a given ξ belongs to RN_ϵ. Let ϵ be a positive real number. Define the set $\mathcal{R}_\epsilon = \{x \in \mathbb{C} \mid |S(x)| \le \epsilon\sigma(x)\}$. The following useful result relates RN_ϵ and $\mathcal{R}_\epsilon$.

Proposition 8. *It holds that* $\mathrm{RN}_\epsilon = \mathcal{R}_\epsilon$*.*

Proof. If $\xi \in \mathrm{RN}_\epsilon$ then there exist ϵ_i, $|\epsilon_i| \le \epsilon$, such that $\sum_{i=1}^n \frac{a_i}{\xi - b_i}(1+\epsilon_i) - 1 = 0$. This implies that $S(\xi) = -\sum_{i=1}^n \frac{a_i}{\xi-b_i}\epsilon_i$, whence $|S(\xi)| \le \sigma(\xi)\epsilon$, that is $\xi \in \mathcal{R}_\epsilon$.

If $\xi \in \mathcal{R}_\epsilon$, then for $\eta = S(\xi)$ one has $|\eta| \le \epsilon\sigma(\xi)$. Set

$$\delta_i = -\frac{\xi - b_i}{a_i}\left|\frac{a_i}{\xi - b_i}\right|\frac{\eta}{\sigma}(\xi)$$

and find that $|\delta_i| \le \left|\frac{\eta}{\sigma(\xi)}\right| \le \epsilon$, moreover,

$$\sum_{i=1}^n \frac{a_i}{\xi - b_i}\delta_i = -\frac{\eta}{\sigma(\xi)}\sum_{i=1}^n \left|\frac{a_i}{\xi - b_i}\right| = -\sigma(\xi).$$

Whence, $\widetilde{S}(\xi) = 0$ with $\widetilde{S}(x) = \sum_{i=1}^n \frac{a_i}{x-b_i}(1+\delta_i) - 1$. That is $\xi \in \mathrm{RN}_\epsilon$. □

In the actual computations in floating point arithmetic, due to the round-off errors, we cannot check if $\xi \in \mathcal{R}_\epsilon$. What we can do is to test the condition $x \in \widetilde{\mathrm{RN}}_{\epsilon,u}$ where

$$\widetilde{\mathrm{RN}}_{\epsilon,u} = \{x \in \mathbb{C} : \quad |\mathrm{fl}_u\,(S(x))| \leq \epsilon\sigma(x)\}.$$

In view of Proposition 8 and Corollary 4, we find that for a given ϵ and a given machine precision u the following property holds

Proposition 9. *Let ϵ be such that $\epsilon \geq (1+u)u\kappa_n$. Then*

$$\begin{gathered}\mathrm{RN}_{\epsilon/(1+u)-u\kappa_n} \subseteq \widetilde{RN}_{\epsilon,u} \subseteq \mathrm{RN}_{\nu(1)\epsilon+u\kappa_n}\\ \widetilde{RN}_{(1-u)(\epsilon-u\kappa_n),u} \subseteq \mathrm{RN}_\epsilon \subseteq \widetilde{RN}_{(1+u)(\epsilon+u\kappa_n),u}\end{gathered}$$

Proof. Concerning the inclusion $\mathrm{RN}_\epsilon \subseteq \widetilde{\mathrm{RN}}_{(1+u)(\epsilon+\kappa_n u),u}$, if $x \in \mathrm{RN}_\epsilon$, then by Proposition 8 $|S(x)| \leq \epsilon\sigma(x)$. Therefore, from Corollary 4 one has $|\mathrm{fl}_u\,(S(x))| \leq (1+u)(\epsilon+u\kappa_n)\sigma(x)$. Whence we deduce that $x \in \widetilde{\mathrm{RN}}_{(1+u)(\epsilon+\kappa_n u),u}$.

Concerning the inclusion $\mathrm{RN}_\epsilon \subseteq \widetilde{\mathrm{RN}}_{(1+u)(\epsilon+u\kappa_n),u}$, we find that if $x \in \widetilde{\mathrm{RN}}_{\epsilon,u}$ then $|\mathrm{fl}_u\,(S(x))| \leq \epsilon\sigma(x)$. Therefore, in view of Corollary 4, we deduce that $|S(x)| \leq (\frac{1}{1-u}\epsilon + \kappa_n u)\sigma(x)$. Since $1/(1-u) = \nu(1)$, the proof of this inclusion is completed. The remaining inclusions follow from these ones. $\square$

It is interesting to point out that if x satisfies the implementable stop condition (14) then $x \in \widetilde{\mathrm{RN}}_{\epsilon,u}$ with $\epsilon = \kappa_n u\frac{(1+2u)(1-u)}{1-2u}$ so that, in view of Proposition 9 one finds that

$$\mathrm{RN}_{\nu(1)\nu(2)u^2\kappa_n} \subseteq \widetilde{\mathrm{RN}}_{\kappa_n u,u} \subseteq \mathrm{RN}_{\nu(2)u\kappa_n}. \tag{15}$$

This property extends to the case of secular equations a similar property valid for polynomials and proved in [5]. The advantage of secular equations is that k_n has a logarithmic growth with respect to n, whereas for polynomials k_n grows linearly with n.

We may conclude with the following important fact

Fact 10. Applying any algorithm that relies on the halt condition (14) and runs with a floating point arithmetic with machine precision u, provides approximation to the roots of $S(x)$ which are the exact roots of secular equations with the coefficients perturbed by a relative error at most $\frac{2\kappa_n u}{1-2u} \doteq 2\kappa_n u$.

4 Representing equivalent secular functions on different sets of nodes

Multiplying $S(x)$ by the polynomial $\Pi(x) = -\prod_{i=1}^n (x-b_i)$ provides the monic polynomial of degree n $p(x) = \Pi(x)S(x)$. From this equation we deduce the following relation between $p(x)$ and the coefficients a_i of $S(x)$.

$$a_i = \frac{p(b_i)}{\prod_{j=1,\, j\neq i}^n (b_i - b_j)} \tag{16}$$

Given the secular function $S(x)$ defined in (1), and given a new set of nodes $\tilde{b}_1, \ldots, \tilde{b}_n$, we may wish to construct an equivalent function using the new set of nodes, i.e., to find $\tilde{a}_1, \ldots, \tilde{a}_n$ such that $\widetilde{S}(x) = \sum_{i=1}^n \frac{\tilde{a}_i}{x - \tilde{b}_i} - 1$ has the same roots as $S(x)$. This condition can be restated as an equality between the polynomials associated with $S(x)$ and $\hat{S}(x)$, i.e., $\prod_{j=1}^n (x - b_i)S(x) = \prod_{j=1}^n (x - \tilde{b}_i)\widetilde{S}(x)$. In terms of the coefficients, we obtain

$$\tilde{a}_i = (\tilde{b}_i - b_j) \prod_{j=1,\, j\neq i}^{n} \frac{\tilde{b}_i - b_j}{\tilde{b}_i - \tilde{b}_j} (\sum_{k=1}^{n} \frac{a_k}{\tilde{b}_i - b_k} - 1). \tag{17}$$

This equation provides a tool for constructing the equivalent function $\widetilde{S}(x)$ in a different set of nodes with the overall cost of $O(n^2)$ operations.

It is possible to estimate the working precision of the floating point computation which guarantees a relative error in the actually computed coefficients $\tilde{a}_i$ bounded by a given ϵ. This estimate can be performed during the computation.

In the case where a few b_i's, say k, have been changed it is possible to give a different expression for the coefficients $\hat{a}_i$ which is much less expensive than (17). More specifically we can suppose, without loss of generality, that $b_i \neq \hat{b}_i$ only for $i = 1, \ldots, k$. We have, for every $i = k+1, \ldots, n$:

$$\hat{a}_i = a_i \prod_{j=1}^{k} \frac{b_i - b_j}{\hat{b}_i - b_j}$$

while the formula (17) can be used for $i = 1, \ldots, k$. It is easy to show that this "partial" regeneration can be implemented with $O(kn)$ arithmetic operations.

4.1 Matrix formulation and Gerschgorin disks

Secular equations are closely related to matrix eigenvalue problems. Here we recall some known properties and we refer the reader for more details to [22, 10, 14] and to the references cited therein.

Consider the matrix

$$A = D - \mathbf{a}\mathbf{e}^T \tag{18}$$

where $D = \operatorname{diag}(b_1, \ldots, b_n)$, $\mathbf{a} = (a_i)$, and $\mathbf{e}$ is the vector with components equal to 1. Observe that $(xI - A) = (xI - D)(I - (xI - D)^{-1}\mathbf{a}\mathbf{e}^T)$ so that

$$p(x) = \det(xI - A) = \prod_{i=1}^{n} (x - b_i)(1 - \mathbf{e}^T (xI - D)^{-1}\mathbf{a}) = \Pi(x)S(x)$$

where $\Pi(x) = -\prod_{i=1}^n (x - b_i)$. Therefore, the eigenvalues of A coincide with the roots of the secular function $S(x)$. This property enables us to provide a set of inclusion disks for the roots of $S(x)$, just by applying the Gerschgorin theorem either to A or to A^T. We recall that in view of the Gerschgorin theorem, the set of eigenvalues of A is contained in the union of the Gerschgorin disks $B(b_i - a_i, r_i)$ of center $b_i - a_i$ and radius $r_i = (n-1)|a_i|$, $i = 1, \ldots, n$. Moreover, any connected component of this union formed by k disks contains exactly k eigenvalues of A. That is they form a set of inclusion disks. For the sake of simplicity, we use a different formulation of this theorem where the Gerschgorin disks are replaced by the disks $G_i = B(b_i, n|a_i|)$ of center b_i and radius $n|a_i|$. It is easy to show that these disks still form a set of inclusion disks.

Given a set of approximations $\tilde{b}_1, \ldots, \tilde{b}_n$ to the roots of $S(x)$, it is possible to construct the equivalent function $\widetilde{S}(x)$ in the new set of nodes $\tilde{b}_1, \ldots, \tilde{b}_n$ by means of (17), with a sufficient working precision which guarantees a relative error in the computed $\tilde{a}_i$ bounded by u. This way, we find that the disks $B(\tilde{b}_i, \tilde{r}_i)$, $\tilde{r}_i = n|\tilde{a}_i|(1+u)$, $i = 1, \ldots, n$, form a set of inclusion disks.

The following result relates Gerschgorin disks and the root-neighborhoods.

Proposition 11. *If $x \in \mathrm{RN}_\epsilon(S)$ then there exists k such that $|x - b_k| \le n\,|a_k|\,(1+\epsilon)$. In particular, the union of the Gerschgorin disks $B(b_i, r_i)$, $r_i = n\,|a_i|\,(1+\epsilon)$ contains $\mathrm{RN}_\epsilon(S)$.*

Proof. If $x \in \mathrm{RN}_\epsilon(S)$ then $\sum_{i=1}^n \frac{a_i}{x-b_i}(1+\epsilon_i) - 1 = 0$ with $|\epsilon_i| \le \epsilon$. Let k be such that $|a_k| = \max_i \left|\frac{a_i}{x-b_i}\right|$ and deduce that $1 \le n\left|\frac{a_k}{x-b_k}\right|(1+\epsilon)$, whence $|x - b_k| \le n\,|a_k|\,(1+\epsilon)$. □

Observe that any connected component of the union of the Gerschgorin disks contains one or more connected components of the root-neighborhood and the number of disks forming the component enable us to count the number of roots of any secular function in the subset of the secular neighborhood.

Observe also that, if x satisfies the stop condition (14) then, in view of (15), $x \in \mathrm{RN}_{\nu(2)u\kappa_n}$. That is there exists k such that the disk $B(b_k, r)$ contains a root of $S(x)$ where $r = a_k(1 + \sigma(x)(2\kappa_n \frac{u}{1-2u}))$.

Another inclusion result can be obtained by using the property that for a given polynomial $p(x)$ and for a given ξ there exists a root of $p(x)$ in the disk $B(\xi, r)$ for $r = n|p(\xi)/p'(\xi)|$, see for instance [21]. Sometimes this bound can be more accurate than the one obtained by the Gerschgorin disks. However, applying this inclusion to the set of nodes $b_1, \ldots, b_n$ provides disks which, in general, do not form a set of inclusion disks. By relying on the relation $p(x) = \Pi(x)S(x)$ one finds that

$$p(x)/p'(x) = \frac{S(x)}{S(x)\sum_{i=1}^n \frac{1}{x-b_i} + S'(x)} \tag{19}$$

which is computable in $O(n)$ operations.

Equation (19) provides the Newton correction which is used not only in the Newton iteration but also in the Ehrlich-Aberth iteration described in the next Section 6. This equation cannot be applied in this form if $x = b_k$ for some k. In this case the following alternative can be used

$$p(b_k)/p'(b_k) = \frac{a_k}{\sum_{i=1,\, i\ne k}^n \frac{a_i + a_k}{b_k - b_i} - 1}. \tag{20}$$

It is convenient to complement condition (14) with the additional test

$$|p(x)/p'(x)| \le u|x|. \tag{21}$$

In fact, if this condition is satisfied, then there is a root in the disk of center x and radius $r = n|p(x)/p'(x)|$ that is x approximates this root with a relative error at most $r/|x| \le nu$. On the other hand, if (21) is satisfied then the Newton correction is so small that it cannot increase the precision of the new approximation.

5 Perturbation result

In this section we analyze the variation of a root ξ of $S(x)$ when the coefficients a_i are modified by a relative perturbation ϵ.

5.1 Estimating the conditioning of a root

Let j be an integer such that $1 \le j \le n$. Consider the perturbed secular function $\hat{S}(x) = \sum_{i=1}^{n} \frac{\hat{a}_i}{x-b_i} - 1$ where $\hat{a}_i = a_i$ for $i \neq j$ and $\hat{a}_j = a_j + a_j\epsilon_j$, $|\epsilon_j| \le \epsilon$.

Let ξ be a root of $S(x)$, and $\hat{\xi}$ the root of $\hat{S}(x)$ closest to ξ so that

$$\begin{cases} \sum_{i=1}^{n} \frac{a_i}{\xi-b_i} - 1 = 0, \\ \sum_{i=1}^{n} \frac{\hat{a}_i}{\hat{\xi}-b_i} - 1 = 0. \end{cases}$$

Setting $\delta_\xi := \hat{\xi} - \xi$, and subtracting both sides in the above system yields

$$\left(\sum_{i=1}^{n} \frac{a_i}{(\xi - b_i)(\hat{\xi} - b_i)} \right) \delta_\xi - \frac{a_j\epsilon_j}{\hat{\xi} - b_j} = 0.$$

From this relation we can easily obtain the explicit expression for the fraction $\frac{\delta_\xi}{\epsilon_j}$ that represents exactly the variation of the root ξ when the coefficient a_j is perturbed. More precisely we have

$$\frac{\delta_\xi}{\epsilon_j} = \frac{a_j}{(\hat{\xi} - b_j)\left(\sum_{i=1}^{n} \frac{a_i}{(\xi-b_i)(\hat{\xi}-b_i)}\right)}, \tag{22}$$

so that, since $\hat{\xi} - b_i = \xi - b_i + \delta_\xi$, we arrive at the following estimate:

$$\left|\frac{\delta_\xi}{\epsilon_j}\right| = \frac{|a_j|}{|\xi - b_j||S'(\xi)|} + O(\delta_\xi^2).$$

In the case where all the coefficients a_j are perturbed by the relative error ϵ_j with $|\epsilon_j| \le \epsilon$, by following the same argument as above, we may prove that

$$|\delta_\xi| \;\dot{\le}\; \left| \frac{\sum_{i=1}^{n} a_i\epsilon_i/(\xi - b_i)}{S'(\xi)} \right| \le \frac{\epsilon\sigma(\xi)}{|S'(\xi)|},$$

where $\dot{\leq}$ denotes inequality up to terms of higher order in the errors. Whence we obtain the following first order bound to the condition number of the root ξ

$$\frac{|\delta_\xi|}{\epsilon} \leq \frac{\sigma(\xi)}{|S'(\xi)|}. \tag{23}$$

5.2 Conditioning and the choice of nodes

Given a secular function $S(x)$, we can consider equivalent functions $\widetilde{S}(x)$ represented with respect to different set of nodes. It is interesting to study how the conditioning of the roots of $\widetilde{S}(x)$ depends on the set of nodes used in the representation.

Assume that for a given k, ξ_k is a simple root of $S(x)$, that is $S'(\xi_k) \neq 0$ and $p'(\xi_k) \neq 0$ where $p(x) = \Pi(x)S(x)$. Equation (23) says that in a first order error analysis the condition number of the root ξ_k is bounded by $\sigma(\xi_k)/|S'(\xi_k)|$. Recall that $\sigma(\xi_k) = \sum_{j=1}^n |\frac{a_j}{\xi_k - b_j}|$ and that $a_j = p(b_j)/\prod_{i=1,\, i\neq j}^n (b_j - b_i)$. Now, let b_i tend to ξ_i for $i = 1, \ldots, n$ and analyze the limit of a_j. If $j \neq k$, the limit of a_j is clearly zero. In fact, in the limit, a_j converges to $p(\xi_j)/p'(\xi_j)$. Therefore in the summation $\sigma(\xi_k) = \sum_{j=1}^n |a_j/(\xi_k - b_j)|$ the only term of interest is the one such that $j = k$. On the other hand, if $j = k$ we may write

$$a_k = \frac{\epsilon_k p'(\xi_k) + O(\epsilon_k^2)}{\prod_{j=1,\, j\neq k}^n (\xi_k - \xi_j + \epsilon_k - \epsilon_j)}, \quad \epsilon_j = \xi_j - b_j,$$

so that

$$|\frac{a_k}{\xi_k - b_k}| = |\frac{a_k}{\epsilon_k}| = |\frac{p'(\xi_k) + O(\epsilon_k)}{\prod_{j=1,\, j\neq k}^n (\xi_k - \xi_j + \epsilon_k - \epsilon_j)}|.$$

If ϵ_j converge to zero then the denominator in the right-hand side of the above expression converges to $p'(\xi_k)$ so that the quantity $|\frac{a_k}{\xi_k - b_k}|$ as well as $\sigma(\xi_k)$ converge to 1. To analyze the condition number $\sigma(\xi_k)/|S'(\xi_k)|$, it remains to study the behavior of $S'(\xi_k)$ when b_i converge to ξ_i. We have $S'(\xi_k) = -\sum_{j=1}^n \frac{a_j}{(\xi_k - b_j)^2}$. Once again, if $j \neq k$ the term $\frac{a_j}{(\xi_k - b_j)^2}$ converges to zero. For $j = k$, we may write

$$\frac{a_k}{(\xi_k - b_k)^2} = \epsilon_k^{-1} \frac{a_k}{(\xi_k - b_k)},$$

where we have already pointed out that $|\frac{a_k}{(\xi_k - b_k)}|$ converges to 1. This means that the ratio $\sigma(\xi_k)/|S'(\xi_k)|$ which express a first order bound to the condition number of the root ξ_k converges to zero.

In other words, for simple roots the condition number decreases to zero when the nodes b_i get close to the roots.

For roots with multiplicity greater than one the analysis is more involved. Let $|\epsilon_j| \leq \epsilon$ and consider the case of a root ξ of multiplicity $m > 1$. For simplicity, assume that $\xi = \xi_1 = \ldots = \xi_m$. In this case $S^{(i)}(\xi) = p^{(i)}(\xi) = 0$ for $i = 1, \ldots, m-1$, $S^{(m)}(\xi), p^{(m)}(\xi) \neq 0$. This way, for $k = 1, \ldots, m$ (16) turns into

$$a_k = \frac{\epsilon_k^m p^{(m)}(\xi) + O(\epsilon_k^{m+1})}{\prod_{j=1,\, j\neq k}^n (\xi - \xi_j + \epsilon_k - \epsilon_j)}, \quad \epsilon_j = \xi_j - b_j.$$

We rewrite the product in the denominator of the above equation as

$$\begin{aligned}\prod_{j=1,\, j\neq k}^{n} (\xi - \xi_j + \epsilon_k - \epsilon_j) &= \prod_{j=1,\, j\neq k}^{m} (\epsilon_k - \epsilon_j) \prod_{j=m+1}^{n} (\xi - \xi_j + \epsilon_k - \epsilon_j) \\ &= \prod_{j=1,\, j\neq k}^{m} (\epsilon_k - \epsilon_j) \left(p^{(m)}(\xi) + O(\epsilon)\right),\end{aligned}$$

so that, since $\epsilon_k - \epsilon_j = b_j - b_k$, one has

$$\frac{a_k}{\xi - b_k} = \frac{\epsilon_k^{m-1}}{\prod_{j=1,\, j\neq k}^{m} (b_j - b_k)} + O(\epsilon) =: \eta + O(\epsilon).$$

If convergence of the nodes $b_1, \ldots, b_m$ to the multiple root ξ is such that the function $\eta = \frac{\epsilon_k^{m-1}}{\prod_{j=1,\, j\neq k}^{m}(b_j - b_k)}$ is bounded from below in modulus, then the same argument used in the case of simple roots applies and we conclude that also the conditioning of multiple roots converges to zero as the nodes converge to the roots. The boundedness from below of $|\eta|$ is satisfied if the nodes $b_1, \ldots, b_m$ are well separated, say, if $b_i = \xi + \epsilon\omega_m^i$, where ω_m is a principal mth root of 1. In fact, in this case one has $|\prod_{j=1,\, j\neq k}^{m}(b_j - b_k)| = \epsilon^{m-1}|\prod_{j=1,\, j\neq k}^{m}(\omega_m^k - \omega_m^j)| = (m-1)\epsilon^{m-1}$.

It is interesting to observe that due to the implicit deflation performed in the Ehrlich-Aberth iteration the convergence of the approximations to multiple roots generally respects the property of good separation of the roots. A nice interpretation of this fact in terms of electric charges which repel each other is given in the original paper by Aberth [1].

In Section 6.1 we present a strategy of relocation of the approximations to the clustered roots of $S(x)$ where the new approximations are equally placed along a circle centered in the multiple root. This way, the condition of boundedness of $|\eta|$ is satisfied.

6 The Ehrlich-Aberth iteration

As already said in the introduction, the main engine to approximate the roots of $S(x)$ that we use is the EA iteration applied to the monic polynomial $p(x) = S(x)\Pi(x)$. This method, designed in [1, 13], has been successfully used in the implementation of MPSolve [4, 5]. There are no theoretical results concerning the global convergence of the Ehrlich-Aberth iteration. However, in practice, convergence always occurs and is quite fast if the choice of the initial approximations is performed in a suitable way.

An effective criterion for choosing initial approximations is reported in [4, 5]. We recall it here in terms of the Pellet theorem [25].

Let $c(x) = \sum_{i=0}^{n} c_i x^i$ be a polynomial of degree n. We recall that the equation $|c_h|x^h = \sum_{i=0,\, i\neq h}^{n} |c_i|x^i$ has one real solution $t_0 > 0$ for $h = 0$, one real solution $s_n > 0$ for $h = n$, either no real positive solutions or two positive solutions $s < t$ otherwise [4].

Let $h_1, \ldots, h_p$ be the values for which the above equation has either one or two positive solutions $s_{h_i} \le t_{h_i}$, $i = 1, \ldots, p$, where we assume $s_0 = 0$, $t_n = \infty$.

Theorem 12 (Pellet theorem)**.** *The open annulus $\{z \in \mathbb{C} : \quad s_{h_i} < |z| < t_{h_i}\}$ contains no roots of $c(x)$ and of any other polynomial $v(x) = \sum_{i=0}^{n} v_i x^i$ such that $|v_i| = |c_i|$. The closed annulus $\{z \in \mathbb{C} : \quad s_{h_{i-1}} \le |z| \le t_{h_i}\}$ contains $h_i - h_{i-1}$ roots of $c(x)$ and of any other polynomial $v(x)$.*

The idea of [4] is to choose initial approximations in the nonempty annuli given in the Pellet theorem. However, since computing the real roots s, may be expensive, in [4] it is proposed an almost inexpensive technique, based on the Newton polygon construction, which is resumed in the following

Theorem 13. *Let $(k_i, \log|c_{k_i}|)$, $i = 1, \ldots, q$ be the vertexes of the upper part of the convex hull of the set $\{(k, \log|c_k|), \quad k = 0, \ldots, n\}$. Then $q \ge p$, $\{h_1, \ldots, h_p\} \subseteq \{k_1, \ldots, k_q\}$, moreover, $\{r_1, \ldots, r_{q-1}\} \cap]s_{h_i}, t_{h_i}[= \emptyset$, for $r_i = \left|\frac{c_{k_i}}{c_{k_{i+1}}}\right|^{\frac{1}{(k_{i+1}-k_i)}}$*

The criterion for choosing starting approximation is the following: choose $k_{i+1} - k_i$ approximations equally placed along the circle of center 0 and radius r_i. As a consequence, all the approximations chosen this way belong to the Pellet annuli. The cost of computing the upper part of the convex hull (Newton polygon) with the indices h_i and the radii r_i is just $O(n \log n)$ operations.

6.1 Policy of cluster analysis

The criterion based on the Newton polygon, described above, is used in [5] for accelerating convergence in the case of clustered or multiple roots. Let us recall the following theorem:

Theorem 14 (Marden-Walsh)**.** *Assume that the polynomial $p(x)$, of degree n, has m zeros in the disk $B(c, r)$ and $n - m$ zeros outside the disk $B(c, R)$, where $0 < r < R$. Then, if $(r + R)/r > 2n/m$ then the first derivative $P'(x)$ has a zero in $B(c, r)$.*

Assume that we are given a set of inclusion disks $B_1, \ldots, B_n$. If there is a cluster of m overlapping disks, say $B_1, \ldots, B_m$ which is sufficiently isolated from the remaining disks so that Theorem 14 guarantees the existence of a root ξ of $c^{(m-1)}(x)$ in the cluster, we may compute ξ by applying a few steps of Newton's iteration to $c^{(m-1)}(x)$. Then compute the coefficients of $v(x) = c(x+\xi)$ and apply the Newton polygon technique to $v(x)$ for repositioning m initial approximations in the cluster formed by $B_1, \ldots, B_m$. Accept this set of new approximations only if they belong to the union of the cluster of disks.

Indeed, if there is a zero of multiplicity m in the cluster of disks, then it coincides with the simple zero ξ of the $(m-1)st$ derivative of $c(x)$, this way it can be readily approximated say by Newton's iteration. If there is a tight cluster

Figure 1: Roots of the secular function $S_{3200}(x)$.

of roots in the union of the first m disks, then the Newton polygon technique provides close approximations. In fact, the tighter the cluster, the more efficient the location of the roots provided by the Newton polygon.

7 The software package and the numerical experiments

Algorithm 1 of Section 2 has been implemented in the language C by relying on the theoretical tools presented in the above sections and on the package GMP for multiple precision arithmetic [18]. The software is free and can be downloaded from `http://riccati.dm.unipi.it/mpsolve`. The implementation can exploit the parallel capabilities of the architecture. On multicore computers the software provides the highest performances.

7.1 Numerical experiments

We report here some numerical experiments concerning the application of our software to secular and polynomial equations. Here and hereafter, we refer to our implementation with the term *Secsolve*. In our tests, we compare the CPU time of our algorithm with the time of the software Eigensolve of [15] and of the software MPSolve 2.0 of [5].

The tests have been run in two ways: with and without the use of parallelism in order to allow a more fair comparison with Eigensolve and MPSolve which cannot exploit the advantage of a parallel architecture. Again for a fair comparison, the goal of the computation has been set to approximating all the roots with 10 decimal digits. The tests were run on a computer with two 6-core hyper-threading$^{\mathrm{TM}}$ processors that account for a total of 24 virtual cores.

We take, as a first example, the secular function $S_n(x) = \sum_{i=1}^{n} \frac{(-1)^i}{x-\frac{1}{i}} - 1$. The CPU time reported in Table 1 clearly shows that in this case the complexity of our algorithm grows quadratically with the degree. The acronym "mt" denotes the CPU time for the multi-threading (parallel) implementation of Secsolve. The same table shows that the acceleration due to the parallel implementation is quite good.

Among the test polynomials, we treat separately the cases of polynomials having roots with an almost uniformly distributed numerical conditioning and the case of polynomials having a few ill conditioned roots. The former, is the case where in principle our algorithm provides the best performances, the latter is the case where the MPSolve strategy should be most suited. We consider also the case of sparse polynomials having few nonzero coefficients. MPSolve takes a great advantage from this feature since it relies on the sparse Horner rule, while

Degree	Secsolve	Secsolve (mt)
200	0.22	0.08
400	0.78	0.15
800	4.00	0.48
1600	15.90	2.18
3200	63.53	8.26

Table 1: Timings (in seconds) for computing the roots of the secular function $S_n(x) = \sum_{i=1}^{n} \frac{(-1)^i}{x-\frac{1}{i}} - 1$.

Eigensolve and Secsolve cannot. The set is completed relying on the polynomial testsuite of MPSolve.

7.2 Polynomials with almost all ill-conditioned roots

In this class of polynomials we considered the following cases:

Mandelbrot polynomials: These polynomials have integer coefficients. They can be obtained by a recurrence relation in the following way where $p_k(x)$ has degree $2^k - 1$:

$$\begin{cases} p_0(x) = 1 \\ p_{k+1}(x) = xp_k^2 + 1, \quad k = 0, 1, \ldots. \end{cases}.$$

The roots of these polynomials describe the Mandelbrot set, the roots of the polynomial of degree 1023 are shown in Figure 2.

Figure 2: Roots of the Mandelbrot polynomial of degree 2047

Partition polynomials: In these polynomials, the coefficient of x^k is a positive integer that coincides with the number of different ways in which the integer k can be decomposed as a sum of non negative integers. The roots of these polynomials have a particular pattern, shown in Figure 3. They have been

Figure 3: The roots of the partition polynomial of degree 3200.

analyzed in [9] where the package MPSolve 2.0 was used to solve the partition polynomial of degree 70.000 in about 30 days of CPU time. Our algorithm can solve it in less than two hours.

In Tables 2 and 3 we report the CPU time of the different implementations. The smallest time has been typed in boldface, the second smallest time is underlined. One can see that for the polynomials in this class, Secsolve greatly outperforms MPSolve and Eigensolve. Moreover, if parallelism is allowed, the cost in terms of CPU time is strongly reduced. We may also notice that the CPU time, as function of the degree, grows substantially slower for Secsolve than for the other two implementations. The same observation is valid for partition polynomials.

Degree	MPSolve 2.2	Eigensolve	Secsolve	Secsolve (mt)
255	2.880	<u>2.250</u>	**0.660**	0.160
511	31.890	<u>20.050</u>	**4.74**	0.740
1023	458.040	<u>229.370</u>	**36.91**	4.510
2047	11158.7	<u>3860.1</u>	**313.44**	33.33

Table 2: Timings for the solution of Mandelbrot polynomials.

Degree	MPSolve 2.2	Eigensolve	Secsolve	Secsolve (mt)
800	<u>5.5</u>	36.9	**2.11**	0.46
1600	<u>38.5</u>	390.6	**10.86**	1.77
3200	<u>213.3</u>	7039.0	**53.6**	7.78
6400	<u>1303.8</u>	45953.3	**287.60**	39.67
12800	<u>7845.6</u>	*	**1588.41**	220.93
25600	*	*	*	1194.63
51200	*	*	*	6659.08

Table 3: Timings for the solution of partition polynomials.

Polynomials with roots along curves: Here we take into consideration the polynomials of the MPSolve benchmark which have roots placed along curves, namely: the characteristic polynomials of two suitable Toeplitz matrices, the truncated exponential series, Wilkinson polynomial and orthogonal polynomials like Legendre, Laguerre, Hermite and Chebyshev polynomials.

Also in this case, we may observe the better performances of Secsolve with respect to Eigensolve and MPSolve.

7.3 Polynomials with few ill conditioned roots

For polynomials in this class, like Mignotte-like polynomials the MPSolve approach is more effective, however, the performances of our algorithm are not much inferior. Mignotte-like polynomials are of the kind $p(x) = x^n + (ax+1)^m$ with $n >> m$, $|a| >> 1$. They have $m-n$ well conditioned roots roughly placed along a circle centered at the origin, and m clustered roots placed in a tiny disk centered at $-1/a$.

7.4 Other polynomials

Here we consider the remaining polynomials of the MPSolve benchmark. This set includes polynomials with sparse coefficients, that is such that almost all the coefficients are zero, like `nroots, nrooti`, and `sparse`. MPSolve takes advantage of sparsity since it uses the sparse Horner rule for polynomial evaluation, while Eigensolve and Secsolve, dealing with the secular equation cannot fully

	MPSolve 2.2	Eigensolve	Secsolve	Secsolve (mt)
mig1_100_1.pol	**0.080**	<u>0.210</u>	0.360	0.200
mig1_200.pol	**0.020**	0.670	<u>0.150</u>	0.130
mig1_200_1.pol	**0.200**	1.280	<u>1.020</u>	0.580
mig1_500.pol	**0.210**	8.160	<u>0.640</u>	0.490
mig1_500_1.pol	**0.880**	24.640	<u>2.800</u>	1.990

Table 4: Timings for the solution of polynomials with few ill-conditioned roots

exploit sparsity. However, the CPU time of Secsolve is not much larger than MPSolve, while Eigensolve takes extremely large timings. For the description of the features of the remaining polynomials we refer the reader to the MPSolve web page.

Acknowledgments

The authors wish to thank the anonymous referees for providing useful suggestions that helped to improve the presentation of the paper.

References

[1] O. Aberth, Iteration methods for finding all zeros of a Polynomial simultaneously, *Math. Comp.* 27, 122:339–344, 1973.

[2] A. Amiraslani, P. Lancaster, Rayleigh quotient algorithms for nonsymmetric matrix pencils, Numerical Algorithms, 51(1): 5-22 (2009)

[3] A. Amiraslani, R.M. Corless, L. Gonzalez-Vega, A. Shakoori, 2004. Polynomial algebra by values. Tech. Rep. 04-01, Ontario Research Centre for Computer Algebra.

[4] D.A. Bini, Numerical computation of polynomial zeros by means of Aberth's method, Numer. Algorithms, 13:179–200, 1996

[5] D.A. Bini, G. Fiorentino, Design, analysis and implementation of a multiprecision polynomial rootfinder, Numer. Algorithms, 23:127–173, 2000.

[6] D.A. Bini, L. Gemignani, V. Pan, Fast and stable QR eigenvalue algorithms for generalized companion matrices and secular equations. Numer. Math., 100:373408, 2005.

[7] D.A. Bini, L. Gemignani, V.Y. Pan, Improved initialization of the accelerated and robust QR-like polynomial root-finding. Electron. Trans. Numer. Anal. 17 (2004), 195205.

	MPSolve 2.2	Eigensolve	Secsolve	Secsolve (mt)
chebyshev80.pol	0.110	0.080	**0.060**	0.060
chebyshev160.pol	0.970	0.740	**0.260**	0.120
chebyshev320.pol	9.380	7.020	**1.580**	0.290
exp100.pol	0.110	0.140	**0.060**	0.060
exp200.pol	1.060	1.200	**0.290**	0.080
exp400.pol	10.260	10.700	**6.390**	0.720
hermite80.pol	**0.040**	0.080	0.070	0.050
hermite160.pol	0.550	0.560	**0.180**	0.100
hermite320.pol	5.320	5.500	**1.060**	0.200
laguerre80.pol	0.160	0.090	**0.060**	0.060
laguerre160.pol	1.380	0.720	**0.290**	0.120
laguerre320.pol	14.670	6.940	**6.910**	0.720
legendre80.pol	0.100	0.080	**0.060**	0.060
legendre160.pol	0.970	0.710	**0.260**	0.120
legendre320.pol	8.970	7.020	**1.570**	0.280
toep1_128.pol	0.180	0.370	**0.090**	0.060
toep1_256.pol	1.550	3.040	**0.490**	0.150
toep2_128.pol	0.420	0.480	**0.130**	0.090
toep2_256.pol	4.350	4.170	**0.820**	0.200
wilk40.pol	0.040	0.020	**0.020**	0.040
wilk80.pol	0.300	**0.080**	0.090	0.060
wilk160.pol	2.940	0.690	**0.460**	0.140

Table 5: Timings for the solution of polynomials with roots along curves

[8] D.A. Bini, L. Gemignani, V.Y. Pan, Inverse power and Durand-Kerner iterations for univariate polynomial root-finding. Comput. Math. Appl. 47 (2004), no. 2-3, 447459.

[9] R. P. Boyer and W. M. Y. Goh. Partition polynomials: asymptotics and zeros. Tapas in experimental mathematics, 99–111, *Contemp. Math.*, 457, Amer. Math. Soc., Providence, RI, 2008.

[10] C. Carstensen, Inclusion of the Roots of a Polynomial Based on Gerschgorin's Theorem, *Numer. Math.*, 59:349–360, 1991.

[11] J.J.M. Cuppen, A divide and conquer method for the symmetric tridiagonal eigenvalue problem, Numer. Math., 36(1981), 177–195.

[12] E. Durand, Solutions numériques des équations algébriques, Tome 1: Equations du type F (X) = 0; Racines d'un polynôme, Masson, Paris 1960.

[13] L.W. Ehrlich, A Modified Newton Method for Polynomials, Comm. of ACM, 10, 2 (1967) 107–108.

[14] Elsner, L.: A Remark on Simultaneous Inclusions of the Zeros of a Polynomial by Gershgorin's Theorem. Numer. Math. 21, 425-427 (1973)

[15] S. Fortune, An iterated eigenvalue algorithm for approximating the roots of univariate polynomials, Journal of Symbolic Computation 33, 5, 2002, 627–646.

[16] D.R. Fuhrmann, An algorithm for subspace computation with application in signal processing, SIAM J. Matrix Anal. Appl. 9,2(1988), 213–220.

[17] G.H. Golub, Some modified matrix eigenvalue problems. SIAM Review, 15:318334, 1973.

[18] GMP: The GNU Multiple Precion Arithmetic Library. `http://gmplib.org/`

[19] M. Gu, S. Eisenstat, A divide-and-conquer algorithm for the symmetric tridiagonal eigenproblem, SIAM J. Matrix Anal. Appl. 16(1995) 172–191.

[20] N.J. Higham, Accuracy and Stability of Numerical Algorithms, Second Edition. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002.

[21] P. Henrici, *Applied and Computational Complex Analysis,* Vol. 1, Wiley, 1974.

[22] Householder,.A.S.: The Theory of Matrices in Numerical Analysis. New York: Blaisdell 1964

[23] A. Melman, Numerical solution of a secular equation, Numer Math. 69(1995), 483–493.

[24] R.G. Mosier, Root Neighborhoods of a Polynomial, Math. Comp., 47 (1986) 265–273.

[25] A.E. Pellet, Sur un mode de séparation des racines des équations et la formule de Lagrange, Bulletin des Sciences Mathématiques, (2), vol 5 (1881), pp. 393–395.

[26] P. Tilli, Convergence conditions of some methods for the simultaneous computation of polynomial zeros, Calcolo, 35(1998), 3-15.

[27] R. Vandebril, M. Van Barel, N. Mastronardi, Matrix Computations and Semiseparable Matrices. The Johns Hopkins University Press, Baltimora, MD, 2008.

	MPSolve 2.2	Eigensolve	Secsolve	Secsolve (mt)
chrma86.pol	0.200	0.090	**0.070**	0.060
chrma342.pol	19.410	4.440	**2.540**	0.390
chrma_d84.pol	0.260	**0.060**	0.080	0.060
chrma_d340.pol	29.710	4.460	**3.380**	0.510
chrmc343.pol	32.000	5.000	**3.490**	0.780
chrmc_d171.pol	2.500	0.590	**0.470**	0.170
chrmc_d683.pol	493.230	37.680	**28.090**	3.230
curz80.pol	0.090	**0.050**	0.060	0.050
curz160.pol	0.710	0.360	**0.220**	0.100
easy400.pol	**0.200**	1.720	**0.200**	0.070
easy800.pol	0.760	10.670	**0.710**	0.140
easy1600.pol	3.040	77.910	**2.750**	0.370
easy3200.pol	**20.740**	551.540	24.340	7.110
geom2_40.pol	**0.020**	0.540	**0.020**	0.030
geom3_20.pol	**0.010**	0.020	**0.010**	0.020
geom3_80.pol	**0.020**	2.620	0.030	0.040
geom4_80.pol	**0.030**	2.720	0.040	0.030
kats8.pol	28.870	**2.310**	2.440	0.410
kir1_10.pol	0.140	**0.070**	0.100	0.060
kir1_10_mod.pol	0.050	0.040	**0.030**	0.050
kir1_20.pol	0.640	**0.190**	0.230	0.150
kir1_20_mod.pol	0.180	0.130	**0.080**	0.070
kir1_40.pol	6.850	**1.570**	1.640	0.450
kir1_40_mod.pol	1.180	1.040	**0.330**	0.160
mult2.pol	**0.030**	0.040	0.050	0.060
nrooti800.pol	**0.100**	29.330	0.120	0.070
nrooti1600.pol	**0.330**	245.930	0.400	0.180
nrooti3200.pol	**1.600**	*	1.620	0.720
nroots800.pol	**0.070**	8.380	0.170	0.070
nroots1600.pol	**0.340**	58.500	0.560	0.210
nroots3200.pol	**1.310**	404.170	2.240	0.740
sendra80.pol	0.450	**0.080**	0.090	0.080
sendra160.pol	5.120	**0.540**	0.630	0.220
sendra320.pol	74.530	**4.490**	4.830	0.710
sparse200.pol	**0.020**	0.260	0.030	0.040
sparse400.pol	**0.040**	1.420	0.060	0.040
sparse800.pol	**0.120**	8.430	0.150	0.090
sparse6400.pol	**6.990**	*	10.820	6.800
spiral20.pol	0.100	**0.010**	0.060	0.060
spiral25.pol	0.200	**0.030**	0.080	0.080
spiral30.pol	14.830	**0.050**	0.160	0.100

Table 6: Timings for the solution of other polynomials