UNIVERSITÀ DEGLI STUDI DI PISA

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea Magistrale in Matematica

Tesi di Laurea

A ROOTFINDING ALGORITHM FOR POLYNOMIALS AND SECULAR EQUATIONS

Relatore:
Prof. DARIO A. BINI

Candidato:
LEONARDO ROBOL

Controrelatore:
Prof. LUCA GEMIGNANI

ANNO ACCADEMICO 2011 – 2012

CONTENTS

INTRODUCTION

This thesis deals with algorithmic issues related to the numerical solution of secular equations. A secular equation is an equation of the kind

$$S(x) = 0, \quad S(x) = \sum_{i=1}^{n} \frac{a_i}{x - b_i} - 1$$

where a_i, b_i are complex numbers and where, for simplicity, we assume that $a_i \neq 0$ and $b_i \neq b_j$ for $i \neq j$.

Secular equations are encountered in different guises in many problems of numerical analysis and scientific computing.

CONSTRAINED EIGENVALUES PROBLEMS Consider the problem of computing the biggest eigenvalue of a symmetric matrix A subject to a linear condition $c^T x = 0$ where c is given and x is the eigenvector relative to λ. In [GGvM89] Golub shows how it's possible to transform this problem in a secular equation.

RANK ONE PERTURBATIONS Secular equations arise in the problem of finding the eigenvalues of symmetric matrix with a rank one perturbation. If we suppose known the eigendecomposition of the starting matrix, the problem of finding the eigenvalues of the perturbed matrix can be solved by finding the roots an appropriate secular equation as shown in [BNS78].

MINIMIZATION PROBLEMS There is an interesting set of least square problems that can be formulated in terms of secular equations. Some important examples are the total least square problems (analyzed in [GVL80]), the least square problems with quadratic constraint, studied in [Gan80] and the regularized truncated total least square problem studied in [FGHO97].

DIVIDE AND CONQUER Eidelman has just presented in [EH12] a divide and conquer algorithm for the computation of the eigendecomposition of quasi-separable matrices that requires, at each step, the solution of a complex secular equation.

Generally, secular equations are formulated in the real domain and the most important applications are the ones where they have real solutions. However, the interest for

the general case is alive and is also motivated by the fact that any polynomial equation can be reduced to a secular equation. This way, any effective algorithm for solving secular equations provides algorithmic advances for the polynomial root-finding problem. In fact, computing roots of polynomial is one of the most ancient and challenging problems in mathematics and has many relevant applications in the solution of certain industrial problems of robot design.

From one hand, the goal of this thesis is to collect theoretical and computational tools which are useful to design secular root-finders with specific features. In particular, we are interested in root-finders which can provide approximations to the secular roots up to any guaranteed accuracy.

On the other hand we aim to arrive at the implementation of a software package for the guaranteed multiprecision solution of secular equations which relies on specific algorithmic strategies and that can exploit the parallel processing features provided by the currently available hardware.

The framework on which we attack the problem is that of numerical computations. In other words, all our algorithms are performed in *floating point arithmetic*. We use, as long as possible, the standard 53-bit IEEE floating point arithmetic which is the fastest currently available arithmetic. When it is needed, we switch to the multiprecision floating point arithmetic where the number of digits, and consequently the working precision, is tuned according to the need of the current computation. In fact, we rely on the GMP package for multiprecision arithmetic implemented by the GNU project. In our approach we avoid to use symbolic computations. The motivation of this fact is that symbolic computations often lead to the growth of the number of digits which is generally not under control. Whereas floating point arithmetic provides an arithmetic framework with uniform cost. This advantage is paid by the presence of round-off errors which, in our case, can be kept under control with a rigorous rounding error analysis. However, we do not exclude that in certain cases, a symbolic preprocessing can improve the efficiency of our algorithm. We leave this issue to our future analysis.

Due to the kind of approach that we have adopted, we need to apply all the numerical tools related to rounding error analysis, like forward and backward analysis, numerical conditioning, perturbation theorems, evaluation of error bounds, *a priori* and *a posteriori* error bounds.

For this reason, we rely both on *the polynomial formulation* of our problem — in fact $S(x)\prod_{i=1}^{n}(x-b_i)$ is a polynomial — and on the *matrix formulation* where the secular roots can be viewed as the eigenvalues of a diagonal plus a rank-one matrix.

Concerning the former formulation, we rely on some key theorems like Rouché, Marden-Walsh, and Pellet theorems, together with the Newton polygon construction and the recent results related to tropical polynomials. Also some classical inclusion theorems related to the Newton correction are applied.

Concerning the latter, we rely on Gerschgorin theorem and the perturbation results of matrix eigenvalues, in particular the Bauer-Fike theorem. The theory of root-neighborhood valid for polynomial roots is extended to secular equation in a straightforward way.

As main engine for improving some available approximations of the roots of polynomials, we use an iteration discovered independently by Ehrlich, Aberth and Börsch-Soupan. This kind of iteration, already experimented successfully in the package `MPSolve`, enables one to refine "simultaneously" a set of approximations and generates a sequence of n-tuples which locally converges to the n-tuple of the roots of the polynomial. Local convergence speed is very high: for simple roots is of the third order. Concerning global convergence, no theoretical results are known so far. However, also no counterexample of non-convergence is known and from the practice of computation, convergence always occurs in a few steps when the starting approximations are chosen with suitable criteria. Here, we adjust this iteration to solving the secular equation. We report also about other techniques which can be used for the simultaneous approximation like the Durand-Kerner or Weierstrass method.

In our approach, we adopted two different algorithmic strategies: the `MPSolve` "philosophy" and the `eigensolve` technique. The latter has been suitably modified in order to speed up the computation. In fact, our package has a switch which enables the user to choose between the two different strategies.

The `MPSolve` strategy relies on the following ideas:

RELATIVE ERROR ANALYSIS In `MPSolve` the error is always estimated by using relative error analysis, instead of absolute error. This seems to be more effective when performing floating point computations.

ADAPTIVITY Instead of using the working precision needed for the worst possible input, `MPSolve` follows an adaptive pattern. The computation starts in standard IEEE floating point and increases the working precision *only when necessary* and *only for the roots that need it*. This allows to obtain a fine tuned algorithm that does not waste computational effort when not necessary.

IMPLICIT DEFLATION Another general rule that `MPSolve` follows is to use the original uncorrupted information at all the stages of the algorithm.

The `eigensolve` strategy works in the following way.

Assume that our goal is to approximate all the roots with d correct bits. A working precision of $w = d + guard$ bits is chosen.

1. Apply any algorithm (say, `MPSolve` or the `QR` iteration) to compute approximations $x_1, \ldots, x_n$ of the roots of the secular equation which are in the 2^{-w} root neighborhood, i.e., that are roots of a secular equation with slightly perturbed coefficients.

2. Represent the secular equation using the approximations delivered at the previous step as nodes, i.e., set $b_i = x_i$ and compute the new a_i. Call $S(x)$ again the equivalent secular function obtained this way. Here, if needed, a higher working precision is used.

3. A stop condition is applied. If the approximations are accurate enough stop the iteration. Otherwise continue from step 1 replacing $S(x)$ with the new secular function obtained at the previous step.

The difference between the two approaches is that the former aims to use the high precision only when it is really needed. This way, well conditioned roots are computed with low precision while ill conditioned roots, say clustered roots, are computed with high precision, often much larger than the output precision but still not exceeding the bounds provided by the perturbation theorems. The precision is gradually increased only for those roots which cannot be otherwise approximated. This philosophy is like zooming in into a cluster with a more powerful microscope (a higher working precision) and use the more powerful microscope only when it is needed in order to separate very close roots.

In the `eigensolve` approach, the precision of computation is essentially the one requested in the output. In fact all the iterations are performed essentially with the output precision. Higher precision is used only for updating the representation of the secular equation according to nodes which are better approximations to the roots. In this way, the sequence of secular functions generated by this method is such that their roots are better and better conditioned as long as convergence occurs. Even though for the initial secular equation there might be very ill conditioned roots which would have required a higher precision, the secular functions generated in the intermediate steps have the same roots as the original function but their conditioning gets closer to 1 as the process is iterated. At the end of the process, all the digits of the roots, corresponding to the working precision, are correct.

The two approaches use high precision (which is most expensive in terms of CPU time) in two different ways. Observe that the longer the computation in high precision the slower the algorithm. As we will see from the numerical experiments there are cases where one approach is extremely superior to the other one. We are also able to describe the class of polynomials/secular equations where this occurs.

In our implementation, in order to reduce the CPU time, we have slightly modified the strategy of `eigensolve` in the following way. Instead of applying steps 1–3 using the output precision as working precision, we start with the standard IEEE 53-bit floating point arithmetic and apply steps 1–3 until no improvement can be obtained in the current precision. If the approximation delivered in this way fulfill the required precision, the algorithm stops. Otherwise the working precision is increased by doubling the number of digits and the algorithm is applied again.

With this approach we can keep great part of the computation at a lower working precision.

Another important issue that enables us to save CPU time is the representation stage. In fact, in the case some approximations remain unchanged, we may take advantage of this fact by developing suitable low cost formulas for regenerating the equation.

A relevant fact which makes our approach much more effective than the original implementation by S. Fortune in [Foro2] is that our approximation engine, i.e., the Erlich-Aberth method, requires only $O(n)$ storage instead of $O(n^2)$. This makes our method applicable even to polynomials with large degrees.

Finally a software improvement has been obtained by using the techniques of threading which enabled us to exploit efficiently the existence of multi-core processors. The implementation of this technique is not trivial at all and has led to modification in the implementation of the Ehrlich-Aberth iteration. In fact, in order to exploit parallelism, we have been compelled to modify the customary "Gauss-Seidel"–style implementation of the algorithm by partially applying the "Jacobi"–style implementation.

We have applied our algorithm to a wide set of test polynomials taken from the original test polynomials of `MPSolve` and some high degree polynomials that arose in applications.

The results that we obtain are, in some cases, a strong acceleration with respect to the previous `MPSolve` approach and `eigensolve`. This is particularly true, for example, in the case of Mandelbrot polynomials (whose roots lie in the Mandelbrot set) and the partition polynomials.

In particular it is worth to point out that the package `MPSolve` was used in [BGo7] by Boyer and Goh to solve a conjecture on partition polynomials. To arrive at this result

the authors had to solve a polynomial of degree 70.000 having coefficients represented by several megabytes. The time needed for this computation with `MPSolve` was about one month of CPU time. With the software provided in our thesis we reach the same goal in about four hours.

There is still a lot of space for improvements in this work and interesting extensions to the theory presented here. It would be interesting to study how is possible to apply the results of this thesis to polynomials represented in different basis, and to apply some extension of this algorithms to matrix polynomials, that often arise in applications. As an example, see the paper from Mackey D.S., Mackey N., Mehl and Mehrmann [MMMM06] where several applications of matrix polynomials are discussed.

NOTATION

The notation and the acronyms used in the thesis are listed in the following table:

$a \doteq b$	a is equal to b if considering the Taylor expansion truncated to the first order.
$a \dot{\leqslant} b$	a is less or equal to b considering the Taylor expansion truncated to the first order.
$\mathrm{fl}(f(x))$	The result of the floating point evaluation of the function f at x.
$a \leftarrow b$	The value b is assigned to the variable a.
A^T	The transpose of the matrix (or the vector) A.
u	The machine precision. In the case of the standard floating point defined in IEEE754 we have that $u = 2^{-53} \approx 10^{-16}$.
$\lceil f(x) \rceil$	The smallest integer bigger than $f(x)$.
$\mathcal{K}(A)$	The conditioning of A, i.e., $\|A\| \left\|A^{-1}\right\|$ where $\|\cdot\|$ is the appropriate norm for the context.
$\mathrm{RN}_\epsilon(S)$	The root-neighborhood of S relative to the perturbation ϵ. See Section 1.3.2 for the definition.
$O(n^k)$	The big O notation, i.e., $f(n) \in O(n^k)$ if and only if $\frac{f(n)}{n^k}$ and $\frac{n^k}{f(n)}$ are both limited for $n \to \infty$.

1

SECULAR EQUATIONS

1.1 INTRODUCTION TO SECULAR EQUATIONS

DEFINITION 1.1: A *secular equation* of degree n is an equation of the form

$$S(x) = \sum_{i=0}^{n} \frac{a_i}{x - b_i} - 1 = 0 \tag{1.1}$$

where $a_i, b_i \in \mathbb{C}$, $a_i \neq 0$, $i = 1, \ldots, n$, and $b_i \neq b_j$ for $i \neq j$; the coefficients b_i are often called the *nodes* of the secular equation. The rational function $S(x)$ is sometimes called *secular function*.

Observe that the assumptions $a_i \neq 0$ and $b_i \neq b_j$ for $i \neq j$ are no loss of generality. In fact, if one of these two conditions is not satisfied, the secular equation can be rewritten with n replaced by a smaller value and with coefficients satisfying both assumptions.

We refer to the solutions of the secular equation $S(x) = 0$, i.e., the zeros of the secular function $S(x)$, as to the *roots of the secular function* $S(x)$ or also the *roots of the secular equation*.

In this chapter we analyze basic operations with secular equations when working in floating point arithmetic. We rely on the classical theory of rounding error analysis, and we refer the reader to the book [Hig96]. For the sake of completeness, in the Appendix A we report an overview of the classical error bounds for complex floating point arithmetic together with the main theoretical results.

1.2 SECULAR EQUATIONS AND POLYNOMIALS

One of the main reasons of our interest in secular equations is the strict interplay that they have with polynomials.

Let $S(x)$ be a secular function defined by the coefficients a_i and b_i as above. Then the monic polynomial

$$P(x) = -S(x)\prod_{i=1}^{n}(x - b_i) \tag{1.2}$$

has the same roots of S. This way, we may associate with the secular equation (1.1), the polynomial equation $P(x) = 0$, where $P(x)$ is the monic polynomial defined in (1.2).

Conversely, given a set $b_1, \ldots, b_n$ of pairwise different nodes, we may associate with a given polynomial $P(x) = \sum_{i=0}^{n} p_i x^i$ of degree n a secular equation of the form (1.1), provided that the set of nodes does not intersects the set $\{\xi_1, \ldots, \xi_n\}$ of the roots of $P(x)$.

In order to show this, let us introduce the following notation

$$\Gamma_{b_i} = -(\prod_{\substack{j=1 \\ j\neq i}}^{n} b_i - b_j)^{-1} \cdot p_n^{-1}, \quad i = 1, \ldots, n, \tag{1.3}$$

and define

$$a_i = P(b_i)\Gamma_{b_i}, \quad i = 1, \ldots, n. \tag{1.4}$$

It is easy to see that the secular equation $S(x) = 0$, where $S(x)$ is the secular function defined by these a_i and b_i, has exactly the same roots of the original polynomial. In fact, consider the polynomial

$$\widetilde{P}(x) = -p_n S(x)\prod_{i=1}^{n}(x - b_i) = -p_n \sum_{i=0}^{n} a_i \prod_{j=1,\, j\neq i}^{n} (x - b_j)$$

and observe that the difference $Q(x) = P(x) - \widetilde{P}(x)$ is a polynomial of degree at most $n - 1$, since the largest degree terms in $P(x)$ and $\widetilde{P}(x)$ cancel out. Moreover, in view of (1.3) and (1.4), one has $Q(b_i) = 0$ for $i = 1, \ldots, n$, so that $Q(x) \equiv 0$, therefore $P(x) = \widetilde{P}(x)$. In this way, given a set of pairwise different nodes $b_1, \ldots, b_n$ such that $P(b_i) \neq 0$, we may represent a polynomial equation $P(x) = 0$ in terms of a secular equation $S(x) = 0$, just by evaluating the values of $a_1, \ldots, a_n$ by means of (1.4) and (1.3).

We refer to the rational function $S(x)$ as to the *(secular) representation with respect to the nodes* $b_1, \ldots, b_n$ of the polynomial $P(x)$. Another interesting relation between poly-

nomials and secular equations can be formulated in terms of generalized companion matrices. See for example [Car91], [Gol73], [MV95], [Foro2], [For] and [BGP04].

DEFINITION 1.2: Given a monic polynomial $P(x)$ of degree n and n pairwise different nodes b_i we will call *generalized companion matrix* the following matrix:

$$C(P,b) = \begin{bmatrix} b_1 & & \\ & \ddots & \\ & & b_n \end{bmatrix} - \begin{bmatrix} a_1 & \cdots & a_n \\ \vdots & & \vdots \\ a_1 & \cdots & a_n \end{bmatrix} = \operatorname{diag}(b_1,\ldots,b_n) - ea^T \tag{1.5}$$

where $a_i = P(b_i)\Gamma_{b_i}$ and $e = (1,\ldots,1)^T$.

We have the following

THEOREM 1.3: *The characteristic polynomial* $\det(xI - C(P,b))$ *of the matrix* $C(P,b)$ *is exactly the polynomial* $P(x)$*, while the* a_i *and* b_i *are the coefficients of the secular equation associated with* P *on the nodes* b_i*.*

Proof. Let $D = \operatorname{diag}(b_1,\ldots,b_n)$, then $\det(xI - D + ea^T) = \det(xI - D)\det(I - (xI - D)^{-1}ea^T) = \prod_{i=1}^{n}(x - b_i)(1 - \sum_{i=1}^{n}\frac{a_i}{x-b_i})$, where we have used the property $\det(I - uv^T) = 1 - v^T u$, valid for any pair of vectors u, v. □

As we will see in Section 1.5.2, this connection allows us to study the conditioning of the roots of the secular equation $S(x) = 0$ by using elementary tools of numerical linear algebra.

1.3 FLOATING POINT EVALUATION AND ROOT NEIGHBORHOOD

In this section we highlight two important facts about secular equations which are related to each other. The first is that the evaluation of a secular equation can be performed by means of a backward stable algorithm. The second concerns the analysis of the root neighborhood of a secular function $S(x)$, that is, roughly speaking, the set of all the roots of all the secular functions obtained by slightly perturbing the coefficients of $S(x)$.

This analysis is fundamental for our algorithmic purposes and shows how the backward stability is an important and desirable property. Consider the following algorithm for evaluating the secular function S at the point x. This algorithm performs

Algorithm 1 Algorithm for the evaluation of $S(x)$

```
1: procedure EvaluateSecular(x)
2:     s ← 0
3:     for i = 1 : n do
4:         t ← a_i/(x − b_i)
5:         s ← s + t
6:     end for
7:     s ← s − 1
8:     return s
9: end procedure
```

the computation in $2n$ additions and n divisions. Observe that in this algorithm the summation of the terms $a_i/(x - b_i)$ is performed sequentially.

That is, Algorithm 6 reported in the Appendix A is implicitly applied. However, the same summation can be performed by means of Algorithm 7 which relies on a recursive technique. We will refer to these two different algorithmic strategies as the *sequential* and the *recursive* approach, respectively.

According to Theorem A.1, the actual value $\mathrm{fl}(t)$ computed in place of t at each step of this algorithm, when using floating point arithmetic with machine precision u, is given by

$$\mathrm{fl}(t) \doteq \frac{a_i}{x - b_i}(1 + \epsilon_i)$$

with $|\epsilon_i| \leqslant (\epsilon_\pm + \epsilon_\div)$, where $\epsilon_\pm = u$ and $\epsilon_\div = \sqrt{2}\frac{7u}{1-7u} \doteq 7\sqrt{2}u$ are the bounds to the local errors of addition/subtraction and division, respectively. See the Appendix A for more details. Summing all the terms and using the fact that

$$\mathrm{fl}\left(\sum_{i=1}^{n} t_i\right) \doteq t_1(1 + \delta_1) + \cdots + t_n(1 + \delta_n),$$

where $|\delta_i| \leqslant \min\{n - 1, n - i + 1\}u$, if the sequential summation algorithm is used, and $|\delta_i| \leqslant \lceil \log_2 n \rceil u$ if the recursive algorithm is used (compare with (A.4) in the Appendix A), yields the following expression for the value $\mathrm{fl}(S(x))$ obtained by applying Algorithm 1 in the Appendix A to the secular function $S(x)$ in floating point arithmetic:

$$\mathrm{fl}(S(x)) \doteq \Big(\sum_{i=1}^{n} \frac{a_i(1 + \delta_i + \epsilon_i)}{x - b_i} - 1\Big)(1 + \delta) \tag{1.6}$$

where δ is the local error generated in computing the last subtraction, such that $|\delta| \leqslant u$.

Summing up, we conclude with the following result concerning the backward stability of Algorithm 1.

PROPOSITION 1.4: *Algorithm 1 for the evaluation of the secular function*

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

is backward stable. More precisely, denoting $\mathrm{fl}\,(S(x))$ *the value delivered by Algorithm 1 in floating point arithmetic, it holds that*

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

where $|\delta| \leqslant u$, $|\delta_i| \leqslant \kappa_n u$ *and*

$$\kappa_n = \begin{cases} n + 7\sqrt{2} & S(x) \textit{ computed by the sequential algorithm} \\ \lceil \log_2 n \rceil + 7\sqrt{2} + 1 & S(x) \textit{ computed by the recursive algorithm} \end{cases}$$

Proof. It follows from (1.6) in view of the bounds $\epsilon_{\pm}$, $\epsilon_{\div}$, reported in the Appendix A. □

The above result has some useful consequences which are reported in the following

COROLLARY 1.5: *For the values* $S(x)$ *and* $\mathrm{fl}\,(S((x))$ *the following inequalities hold*

$$\begin{aligned} |S(x)| &\dot{\leqslant} (1+u)\,|\mathrm{fl}\,(S(x))| + u\kappa_n \sigma(x), \\ |\mathrm{fl}\,(S(x))| &\dot{\leqslant} (1+u)\,|S(x)| + u\kappa_n \sigma(x), \end{aligned}$$

where $\sigma(x) = \sum_{i=1}^{n} \left| \frac{a_i}{x - b_i} \right|$. *Moreover*

$$\begin{aligned} \frac{\mathrm{fl}\,(S(x)) - S(x)}{S(x)} &\doteq \delta + \frac{1}{S(x)} \sum_{i=1}^{n} \frac{a_i}{x - b_i}\delta_i, \\ \left| \frac{\mathrm{fl}\,(S(x)) - S(x)}{S(x)} \right| &\dot{\leqslant} \left(1 + \frac{\kappa_n \sigma(x)}{|S(x)|} \right) u. \end{aligned}$$

REMARK 1.6: The inequalities provided in the above corollary, valid up to the first order terms in δ, are strict in the sense that they turn to equalities for specific choices of the values δ and δ_i satisfying the conditions $|\delta| = u$, $|\delta_i| = \kappa_n u$.

1.3.1 *Stop condition*

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

Corollary 1.5 provides a mean to implement a stop condition based on the relative error estimate in the computation of $\mathrm{fl}(S(x))$. Observe that, if

$$|S(x)| \leqslant \sigma(x) k_n \frac{u}{1-u} \doteq \kappa_n \sigma(x) u \tag{1.7}$$

then the upper bound to the modulus of the relative error provided in Corollary 1.5 is greater than or equal to one. In this case, there is no guarantee that the computed value $\mathrm{fl}(S(x))$ contains useful information. This way, equation (1.7) can be used as a stop condition for halting the iterations in any secular rootfinder which relies on the information contained in the value $S(x)$ taken at x by the secular function.

Equation (1.7) involves the value of $S(x)$ which in a floating point computation is not available. In fact, the floating point arithmetic provides us the value of $\mathrm{fl}(S(x))$. In view of Proposition 1.4, it holds that $\mathrm{fl}(S(x)) \doteq (1+\delta)S(x) + \sum_{i=1}^{n} \frac{a_i}{x-b_i}\delta_i$ is formed by two terms. The first term, $(1+\delta)S(x)$ is close to zero in a neighborhood of a root of the secular equation. The second one can take a value that is bounded from above in modulus by $\sigma(x)\kappa_n u$, moreover, this bound can be reached by specific values of δ_i. Therefore we deduce the following implementable halting condition

$$|\mathrm{fl}(S(x))| \leqslant \kappa_n \sigma(x) u. \tag{1.8}$$

1.3.2 *Root neighborhoods*

Now we introduce the concept of ϵ–root-neighborhood of $S(x)$ which is closely related to the properties of backward stability introduced in Proposition 1.4.

Definition 1.7: Let ϵ be a fixed positive real number. We call *ϵ–root-neighborhood* of $S(x)$ the set

$$\mathrm{RN}_\epsilon(S) = \left\{x \in \mathbb{C} \,|\, \exists \widehat{a}_i \text{ such that } |a_i - \widehat{a}_i| < \epsilon\,|a_i| \text{ and } \hat{S}(x) = 0\right\}$$

where $\widehat{S}$ is the secular function that has $\hat{a}_i$ and b_i as its coefficients.

Moreover, we call ϵ–*secular-neighborhood* the set

$$SN_\epsilon(x) = \left\{ \widehat{S}(x) = \sum_{i=1}^{n} \frac{\hat{a}_i}{x - b_i} - 1, \quad \hat{a}_i = a_i(1 + \epsilon_i),\ |\epsilon_i| = \epsilon \right\}$$

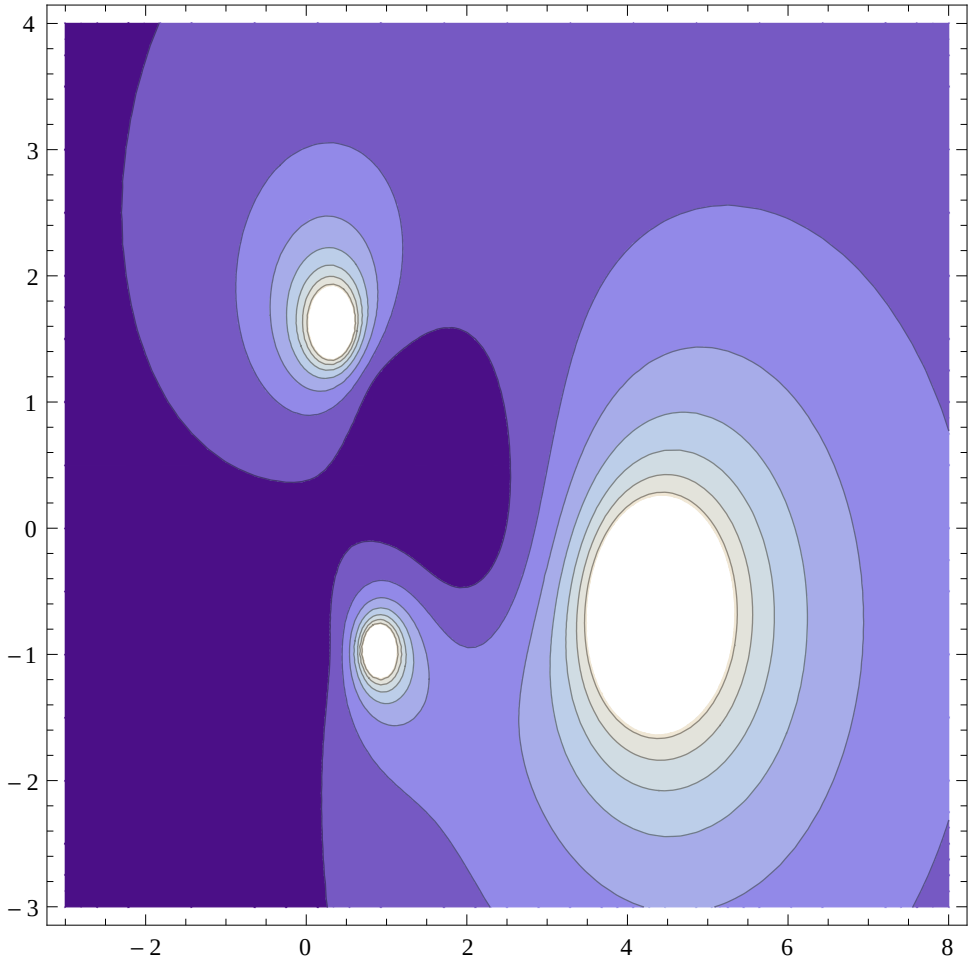


Figure 1.1.: Root neighborhoods of the secular equation $\frac{5}{2(x-2)} - \frac{2}{x-1-i} + \frac{2}{x+i} - 1 = 0$

Recall that the roots of a polynomial are continuous functions of its coefficients. Therefore, since $P(x) = \prod_{i=1} n(b_i - b_j)S(x)$ 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 result

PROPOSITION 1.8: *For any* $\widehat{S}(x) \in SN_\epsilon(S)$ *the roots of* $\widehat{S}(x)$ *belong to the set* $RN_\epsilon(S)$*, moreover, the number of zeros of* $\widehat{S}(x)$ *in any connected component of* $RN_\delta(S)$ *is constant for any* $\delta \leqslant \epsilon$*. In particular, if the set* $RN_\epsilon(S)$ *has* n *connected components then any function* $\widetilde{S}(x) \in SN_\epsilon(S)$ *has one root in each connected component.*

Proof. The first part follows from the definition. The second part follows from the continuity of the roots of $S_\epsilon(x)$. □

The following result relates Gerschgorin discs and the root-neighborhood.

PROPOSITION 1.9: *If* $x \in RN_\epsilon(S)$ *then there exists* k *such that* $|x - b_k| \leqslant n\,|a_k|\,(1+\epsilon)$*. In particular, the union of the Gerschgorin discs* $B(b_i, R_i)$, $R_i = n\,|a_i|\,(1+\epsilon)$ *contains* $RN_\epsilon(S)$.

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

For the sake of notational simplicity, in the following we write RN_ϵ in place of $RN_\epsilon(S)$. This definition aims to clarify the concept of a root in the floating point setting. When trying to approximate a root of the secular equation $S(x) = 0$ we are satisfied if we can find a value x such that $x \in RN_\epsilon$ and ϵ is small enough. In principle, checking the condition $x \in RN_\epsilon$ is not an easy task. However, the following definition provides a way to overcome this difficulty.

DEFINITION 1.10: Let ϵ be a fixed positive real number. We define the set

$$\widehat{RN}_\epsilon = \{x \in \mathbb{C} \mid |S(x)| \leqslant \epsilon\sigma(x)\}$$

Observe that, while checking if $x \in RN_\epsilon$ is computationally unfeasible, verifying that $x \in \widehat{RN}_\epsilon$ can be performed computationally. We can prove the following useful result which relates RN_ϵ and $\widehat{RN}_\epsilon$.

PROPOSITION 1.11: *It holds that*

$$RN_\epsilon = \widehat{RN}_\epsilon$$

Proof. If $x \in RN_\epsilon$ then there exist ϵ_i such that $\sum_{i=1}^{n} \frac{a_i}{x-b_i}(1+\epsilon_i) - 1 = 0$ and $|\epsilon_i| \leqslant \epsilon$. This implies that $S(x) = -\sum_{i=1}^{n} \frac{a_i}{x-b_i}\delta_i$, whence $|S(x)| \leqslant \sigma(x)\epsilon$, that is $x \in \widehat{RN}_\epsilon$.

If $x \in \widehat{\mathrm{RN}}_\epsilon$, then $S(x) = \eta$ and $|\eta| \leqslant \epsilon\sigma(x)$. Set

$$\delta_i = -\frac{x - b_i}{a_i}\left|\frac{a_i}{x - b_i}\right|\frac{\eta}{\sigma}$$

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

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

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

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

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

In view of the above results we find that for a given ϵ and a given machine precision u the following property holds

Proposition 1.12: *Let* $\epsilon > 0$ *then*

$$\mathrm{RN}_\epsilon \subseteq \widetilde{\mathrm{RN}}_{\epsilon+\kappa_n u,u} \subseteq \mathrm{RN}_{\epsilon+2\kappa_n u}$$

moreover

$$\widetilde{\mathrm{RN}}_{\epsilon,u} \subseteq \mathrm{RN}_{\epsilon+\kappa_n u} \subseteq \widetilde{\mathrm{RN}}_{\epsilon+2\kappa_n u,u},$$

Proof. If $x \in \mathrm{RN}_\epsilon$, then in view of Proposition 1.11 $|S(x)| \leqslant \epsilon\sigma(x)$. Therefore, from Corollary 1.5 one has $|\mathrm{fl}\,(S(x))| \dot{\leqslant} \epsilon\sigma(x) + u\kappa_n\sigma(x) = (\epsilon + k_n u)\sigma(x)$. Whence we deduce that $x \in \widetilde{\mathrm{RN}}_{\epsilon+\kappa_n u}$. If $x \in \widetilde{\mathrm{RN}}_{\epsilon,u}$ then $|\mathrm{fl}\,(S(x))| \leqslant \epsilon\sigma(x)$. Therefore, in view of Corollary 1.5, we deduce that $|S(x)| \dot{\leqslant} (\epsilon + \kappa_n u)\sigma(x)$. This completes the proof. □

It is interesting to point out that if x satisfies the implementable stop condition (1.8) then $x \in \widetilde{\mathrm{RN}}_\epsilon$ with $\epsilon = \kappa_n u$ so that, in view of Proposition 1.12 one finds that

$$\mathrm{RN}_{\frac{1}{2}\kappa_n u} \subseteq \widetilde{\mathrm{RN}}_{\kappa_n u,u} \subseteq \mathrm{RN}_{\frac{3}{2}\kappa_n u}$$

This property extends to the case of secular equations a similar property valid for polynomials and proved in [BFoo].

The advantage of secular equations is in the fact that k_n has a logarithmic growth with respect to n if the recursive summation algorithm is applied, whereas for polynomials k_n grows linearly with n. We may conclude with the following important fact

FACT 1.13: *Applying any algorithm that relies on the halt condition* (1.8) *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* $\frac{3}{2}\kappa_n u$. *The set of approximations that we can detect is optimal in the sense that it is in between the two sets* $\mathrm{RN}_{\frac{1}{2}\kappa_n u}$ *and* $\mathrm{RN}_{\frac{3}{2}\kappa_n u}$.

1.4 COMPUTING A NEW REPRESENTATION

As previously noted in Section 1.2, when a secular equation $S(x) = 0$ is given it is possible to obtain a polynomial $P(x)$ with the same roots as $S(x)$ by formally multiplying $S(x)$ by $\prod_{i=1}^{n}(x - b_i)$.

Extending this remark we can produce a set of different secular representations of $P(x)$ simply by changing the nodes b_i. In this section, we will analyze the stability and accuracy of this operation. Both the cases where $P(x)$ is assigned as input polynomial through its coefficients, and when $S(x)$ is assigned in terms of its coefficients a_i and b_i, will be considered.

1.4.1 *Transforming a polynomial equation into a secular equation*

Consider the case where $P(x)$ is given explicitly in terms of its coefficients, or the case where $P(x)$ is implicitly known by means of a "black box" which, given x as input value, provides the value of $P(x)$ by means of a numerically stable algorithm; more specifically suppose that the relative error on the computed value $\mathrm{fl}(P(x))$ is bounded from above by $\kappa_x u$, where κ_x is a known quantity. Let b_k be the nodes for representing the polynomial equation in terms of a secular equation; then, for every $k = 1, \ldots, n$:

$$a_k = -\frac{P(b_k)}{p_n \prod_{\substack{i=1 \\ i \neq k}}^{n}(b_k - b_i)}$$

where p_n is the leading coefficient of $P(x)$. Performing a rounding error analysis we find that

$$\mathrm{fl}\,(b_k - b_i) = (b_k - b_i)(1+\epsilon_i), \quad |\epsilon_i| \leqslant \epsilon_\pm,$$

and multiplying all the terms we finally obtain

$$\mathrm{fl}\left(\prod_{\substack{i=1\\ i\neq k}}^{n}(b_k - b_i)\right) = \left(\prod_{\substack{i=1\\ i\neq k}}^{n}(b_k - b_i)\right)(1+\sum_{\substack{i=1\\ i\neq k}}^{n}(\epsilon_i+\gamma_i)), \quad |\gamma_i| \leqslant \epsilon_*.$$

Since $\epsilon_\pm = u$ and $\epsilon_* \doteq 2\sqrt{2}u$, the relative error on the product is bounded by $nu(1+2\sqrt{2})$.

If we suppose to have a reasonable bound on the error of the polynomial evaluation (and this is true for the Horner scheme used when the coefficients are known) then the whole procedure to compute the new representation is numerically stable. The cost of the operation is bounded by $O(nk+n^2)$ arithmetic operations where k is the cost of a polynomial evaluation. If $k = O(n)$ we have $O(n^2)$ as total cost.

1.4.2 *Changing the nodes of a secular equation*

The whole procedure described in the above section holds valid even when $S(x)$ is known, in place of $P(x)$, if we use the equation

$$P(x) = -p_n S(x)\prod_{i=1}^{n}(x-b_i)$$

for the polynomial evaluation. Since we have already seen that the evaluation of $S(x)$ is backward stable we conclude that even in this case the regeneration of the secular equation on the new nodes is backward stable as well. If we call $\hat{a}_k, \hat{b}_k$ the new coefficients and a_k, b_k the old ones we obtain the following regeneration formula:

$$\hat{a}_k = S(\hat{b}_k)\frac{\prod_{i=1}^{n}(\hat{b}_k - b_i)}{\prod_{\substack{j=1\\ j\neq k}}^{n}(\hat{b}_k - \hat{b}_j)}.$$

As in the previous case the total cost of regeneration is $O(n^2)$.

1.4.3 *Partial regeneration*

There are cases where it is interesting to regenerate a given secular function $S(x) = \sum_{i=1}^{n} \frac{a_i}{x-b_i} - 1$ with respect to a new set of nodes $\widehat{b}_i$, $i = 1, \ldots, n$, where only a few nodes $\widehat{b}_i$ differ from the original nodes b_i. In this circumstances, we may perform the computation at a lower computational cost. In fact, we present a procedure that allows us to perform this regeneration in $O(nr)$ time, where r is the number of indices i such that $b_i \neq \hat{b}_i$. Let R be the set of indices where $b_i \neq \hat{b}_i$. Suppose that $k \notin R$; the formula to compute $\hat{a}_k$ is:

$$\hat{a}_k = \frac{P(\hat{b}_k)}{\prod_{\substack{i=1 \\ i\neq k}}^{n} (\hat{b}_k - b_i)} = \frac{P(b_k)}{\prod_{\substack{i=1 \\ i\neq k}}^{n} (b_k - b_i)} \cdot \prod_{j\in R} \frac{b_k - b_j}{\hat{b}_k - b_j} = a_k \prod_{j\in R} \frac{b_k - b_j}{\hat{b}_k - b_j}.$$

This shows that, for every $k \notin R$, $\hat{a}_k$ is a_k scaled by a coefficient that can be computed in $O(|R|) = O(r)$ time. For the indices in R we can use the usual formula and so the total cost for the regeneration is $O(r \cdot n + r \cdot (n - r)) = O(nr)$. The description of this partial regeneration is reported in Algorithm 2. We may perform a rounding error analysis of this algorithm as usual.

Clearly, for every $k \in R$, the same error analysis already performed in the full regeneration algorithm is still valid. When $k \notin R$, instead, we have that the relative error is bounded from the relative error on the previous coefficient a_k plus the error on the computation of the product $\prod_{j\notin R} \frac{b_k - b_j}{b_k - \hat{b}_j}$. With the usual error analysis we find

Algorithm 2 Partial regeneration

```
1:  procedure PartialRegeneration(b̂_k)
2:      R ← {j = 1, ..., n | b_j ≠ b̂_j}
3:      for k = 1, ..., n do
4:          if k ∉ R then
5:              â_k ← P(b̂_k) / ∏_{j≠k}(b̂_k − b_k)        ▷ Using the standard algorithm
6:          else
7:              â_k ← a_k
8:              for j ∈ R do
9:                  â_k ← â_k (b_k − b_j)/(b̂_k − b̂_j)
10:             end for
11:         end if
12:     end for
13: end procedure
```

that

$$\mathrm{fl}\left(\frac{b_k - b_j}{b_k - \hat{b}_j}\right) = \frac{b_k - b_j}{b_k - \hat{b}_j}(1 + \epsilon_i)$$

with $\epsilon_i \leqslant (2\epsilon_{\pm} + \epsilon_{\div}) =: \epsilon$. Summing all the terms we obtain that at step i the accumulated relative error is $(i-1)\epsilon$, and so we get a bound on the total relative error of

$$\epsilon_{tot} \leqslant |R| \cdot \epsilon.$$

This is a good result since if a few nodes b_i have changed we are able to compute a new representation with a low computational cost and even with a low roundoff error.

REMARK 1.14: In this analysis we have overlooked the fact that the error on $\hat{a}_k$ contains also the previous error on a_k, and this is accumulated at every regeneration step. This may have some implications that cannot be ignored. A straightforward note is that in a multiprecision framework, when the precision of the computation is increased so that we are working with a much smaller machine precision u, the error on a_k will probably become too big with respect to u. In that case the partial regeneration scheme will not be applied, and all the coefficients will have to be regenerated from scratch starting from the uncorrupted original coefficients.

1.5 CONDITIONING OF THE ROOTS

In this section we are interested in studying how the conditioning of the roots changes when we change the secular representation of a polynomial $P(x)$. For this purpose we present some general results on the conditioning of secular equations.

1.5.1 *Conditioning number in the general case*

We start by showing a simple analysis of the conditioning of the roots of a secular equation induced by variations on the coefficients a_i. This is interesting since we have shown in Theorem 1.4 that the evaluation of a secular equation can be performed by a backward stable algorithm. This way, the result computed in floating point arithmetic is the exact value of a secular function with slightly modified coefficients.

For the sake of simplicity we perform our analysis in terms of absolute error. It is almost straightforward to extend this analysis to the case of relative error. Suppose to have $\hat{a}_i = a_i + \epsilon_i$ where $|\epsilon_i| \leqslant \epsilon$, and that x is a solution of

$$\sum_{i=1}^{n} \frac{a_i}{x - b_i} - 1 = 0.$$

We are interested in giving an upper bound (or at least to show that it exists) to the distance $|x - \hat{x}|$ where $\hat{x}$ is the nearest solution of the "perturbed" secular equations

$$\sum_{i=1}^{n} \frac{\hat{a}_i}{\hat{x} - b_i} - 1 = 0.$$

To simplify the analysis suppose that only one of the a_i has been modified, and let us call it a_k. We show two different ways of giving this error bound. In both cases we obtain a first-order evaluation of the error. The first approach is based on explicit error analysis, the second relies on a differential analysis.

Consider the system

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

Setting $\delta x := (\hat{x} - x)$ and $\delta a_k := a_k - \hat{a}_k$ we obtain, by difference and simplification

$$\left(\sum_{i=1}^{n} \frac{a_i}{(x - b_i)(\hat{x} - b_i)} \right) \delta x + \frac{\delta a_k}{\hat{x} - b_k} = 0.$$

From this relation we can easily obtain the explicit expression for the fraction $\frac{\delta x}{\delta a_k}$ that represents exactly the variation of the root x when the coefficient a_k is perturbed. More precisely

$$\frac{\delta x}{\delta a_k} = \frac{1}{(\hat{x} - b_k) \left(\sum_{i=1}^{n} \frac{a_i}{(x - b_i)(\hat{x} - b_i)} \right)}. \tag{1.9}$$

A similar result can be found in a clean way by following the variation of x when we change a_k. More precisely, denote $S_a(x)$ the secular function with the coefficient a in place of a_k so that $S(x) = S_{a_k}(x)$. Assume that x_{a_k} is a simple solution of $S(x)$ and recall that there exists a neighborhood $\mathcal{U}$ of a_k and an analytic function $x_a : \mathcal{U} \to \mathbb{C}$ such that $S_a(x_a) = 0$ for $a \in \mathcal{U}$.

This fact follows from the analog statement on polynomials. Consider a polynomial $P(x) = \sum_{i=0}^{n} p_n x^n$, and let x be a simple root of it.

It can be seen that exists a neighborhood of $(p_0, \dots, p_n)$ in $\mathbb{C}^{n+1}$ where is defined an analytic function $\hat{p}$ to $\mathbb{C}$ such that $P \circ \hat{p} \equiv 0$. Using the connection between secular equations and polynomials we have that for every S_a there exists a polynomial P with the same roots. Moreover, the coefficients of the polynomial are obtained as the image of a continuous function of the coefficients a_i. Considering the preimage of the polynomial neighborhood through this function we obtain a suitable neighborhood of the coefficients a_i that satisfies our requirements.

This implies that

$$\frac{d}{da} S_a(x_a) \equiv 0$$

and in particular

$$\frac{d}{da} S_a(x_a)\bigg|_{a=a_k} = \left(\frac{\partial S_a}{\partial a_k} + \frac{\partial S_a}{\partial x}\frac{dx}{da}\right)\bigg|_{a=a_k} = 0.$$

From the latter equality we can obtain $\frac{dx}{da}$, that is, the value we are looking for, since it is precisely the first order approximation of the variation in x induced by a variation in a_k. Let us denote $\frac{\partial x}{\partial a} = \mathcal{K}_{a_k}$. Computing the derivatives we obtain that

$$\mathcal{K}_{a_k} = \frac{1}{(x - b_k)S'(x)}.$$

Consider again (1.9), and observe that under the assumption of simple root, if $\hat{a}_k \to a_k$ then $x_k \to x$. Therefore, taking the limit in (1.9) yields

$$\frac{\delta x}{\delta a_k} = \frac{1}{(x - b_k)S'(x)}$$

in accordance to the differential analysis. We will see in the next paragraph that a more explicit estimate of the global conditioning can be given knowing the values of the coefficients b_i.

In the case where all the coefficients a_i are perturbed, by following the same argument as above, we may prove that

$$\begin{aligned} |\delta x| \doteq \left|\frac{\sum_{i=1}^n a_i \delta_i/(\tilde{x} - b_i)}{\sum_{i=1}^n a_i/((x - b_i)(\tilde{x} - b_i))}\right| &\dot{\leqslant} \frac{\max_i |\delta_i|\, \sigma(\tilde{x})}{|\sum_{i=1}^n a_i/((x - b_i)(\tilde{x} - b_i))|} \\ &\dot{\leqslant} \frac{\kappa_n \sigma(\tilde{x}) u}{|\sum_{i=1}^n a_i/((x - b_i)(\tilde{x} - b_i))|}. \end{aligned}$$

Moreover, with a differential analysis we get

$$|\delta x| \;\dot{\leqslant}\; \left|\frac{\sum_{i=1}^{n} a_i \delta_i/(\tilde{x}-b_i)}{S'(x)}\right| \;\dot{\leqslant}\; \frac{\max_i |\delta_i|\,\sigma(\tilde{x})}{|S'(x)|} \;\dot{\leqslant}\; \frac{\kappa_n \sigma(x) u}{|S'(x)|}. \tag{1.10}$$

In the case where we have an approximation x to a root that satisfies the stop condition (1.8), we have $x \in \mathrm{RN}_\epsilon$ with $\epsilon = k_n u$. This way, since $x \in \mathrm{RN}_\epsilon$ implies that $\sum_{i=1}^{n} \frac{a_i(1+\delta_i)}{x-b_i} - 1 = 0$ for $|\delta_i| \leqslant \epsilon$, we find that

$$|\delta x| \;\dot{\leqslant}\; \left|\frac{\sigma(x) k_n u}{S'(x)}\right|. \tag{1.11}$$

1.5.2 *Computing the condition number using linear algebra*

It is interesting to exploit the connection between polynomials and secular equations. In fact, we are interested in showing that if some good approximations of the polynomial roots are known then we may exploit this information to obtain a new equation with better conditioned roots. To accomplish this it is sufficient to compute a new representation of the secular equation by using the available approximations as nodes of the representation.

This suggests that a good approximation of the roots of a polynomial could be found by computing a sequence of secular representation of the polynomial itself, where the condition numbers of the roots of this sequence of equations is decreasing.

We can consider the matrix representation of the secular equation that we have already seen in (1.5). Since the roots of the secular equation are the eigenvalue of this matrix, we can obtain an upper bound to the condition number of our problem by computing the one of the eigenvalue problem [1].

Let $M = D - ea^T$ be the matrix associated with our representation, where $D = \mathrm{diag}(b_1, \ldots, b_n)$ is a diagonal matrix with pairwise diagonal entries $b_1, \ldots, b_n$, and a is the vector with nonzero components a_i. In this way, $P(x) = \det(xI - M) = -\prod_{i=1}^{n}(x - b_i)S(x)$ and the eigenvalues of M coincide with the roots of $P(x)$. If the roots of the polynomial $P(x)$ are known then a matrix V that diagonalizes M, i.e., such

1 The variations to the coefficients a_i and b_i induce variations to the associated matrix that are less general that all the possible variations $\delta A \in \mathbb{C}^{n\times n}$. These perturbation are called structured perturbation and the related condition number is called structured condition number. The structured condition number is generally lower than the general condition number.

that $V^{-1}MV$ is diagonal, can be obtained directly. In fact, let λ be eigenvalue of M, with $\lambda \neq b_i$ of any i, and x the corresponding eigenvector such that $Mx = \lambda x$, then

$$(D - ea^T)x = \lambda x \iff (D - \lambda I)x = e(a^T x)$$

whence $x_i = a^T x/(b_i - \lambda)$. It follows that $a^T x \neq 0$, otherwise x would be the null vector. Moreover, normalizing x such that $a^T x = 1$ one has

$$x_i = \frac{1}{b_i - \lambda}.$$

Finally, assuming that b_i is not eigenvalue of M for any i, we can build a matrix V with the eigenvectors of V as columns by simply setting $\lambda = \xi_j$ in the above relation. We obtain:

$$V = (v_{ij}) \qquad \text{where } v_{ij} = \frac{1}{b_i - \xi_j}. \tag{1.12}$$

By using a similar argument, we find that $M^T y = \lambda y$ implies that $(D - \lambda I)y = a(e^T y)$. Since $e^T y \neq 0$, otherwise y would be the null vector, we may normalize y in such a way that $e^T y = 1$. In this way may construct a matrix W whose columns are the eigenvectors of M^T simply by setting $\lambda = \xi_j$ in the above relation. We obtain:

$$W = (w_{ij}) \qquad \text{where } w_{ij} = \frac{a_i}{b_i - \xi_j}. \tag{1.13}$$

From the fact that V and W are the matrices of right and left eigenvectors, it follows that $W^T V = \operatorname{diag}(d)$ is diagonal with

$$d_i = \sum_{j=1}^{n} \frac{a_i}{(b_i - \xi_j)^2} = S'(b_i) \tag{1.14}$$

so that

$$V^{-1} = \operatorname{diag}(d)^{-1} W^T \tag{1.15}$$

We are now interested in studying how the matrix V changes when $b_i \to \xi_i$. If we split $P(b_i) = \prod_{j=1}^{n}(b_i - \xi_j)$ and suppose that $b_i \to \xi_i$ we have that for $i \neq j$

$$w_{ij} = \prod_{k=1}^{n}(b_i - \xi_k) \cdot \left((b_i - \xi_j) \cdot \prod_{\substack{s=1 \\ s \neq i}}^{n} (b_i - b_s) \right)^{-1} = \left(\prod_{\substack{k=1 \\ k \neq j,i}}^{n} \frac{b_i - \xi_k}{b_i - b_k} \right) \frac{b_i - \xi_i}{b_i - b_j} \to 0$$

while in the case where $i = j$:

$$w_{ii} = \prod_{k=1}^{n} (b_i - \xi_k) \cdot \left((b_i - \xi_i) \cdot \prod_{\substack{s=1 \\ s \neq i}}^{n} (b_i - b_s) \right)^{-1} = \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{b_i - \xi_k}{b_i - b_k} \to 1.$$

We can conclude that if b_i are sufficiently near to ξ_i then the matrix that diagonalizes M is close to the identity, and so is well-conditioned. We now recall a basic theorem of numerical analysis regarding a perturbation result for eigenvalues that can be found in [Dem].

THEOREM 1.15 (Bauer-Fike): *Let $A, \delta A$ be matrix in $\mathbb{C}^{n \times n}$ with A diagonalizable, λ an eigenvalue of A and V the matrix that diagonalizes A, i.e., $V^{-1}AV$ is diagonal. Then there exists an eigenvalue μ of $A + \delta A$ such that*

$$|\lambda - \mu| \leqslant \mathcal{K}(V) \cdot \|\delta A\|, \quad \mathcal{K}(V) = \|V\| \left\|V^{-1}\right\|.$$

where $\|\cdot\|$ is any absolute norm.

Theorem 1.15 states that the perturbation of the eigenvalues can be bounded by the condition number of V and the norm of the perturbation of A.

Concluding, we obtain that $\mathcal{K}(V)$ is an upper bound to the conditioning of the roots of a secular equations, too. In view of equations (1.12), (1.13), (1.14), (1.15), we find that in the infinity norm

$$\mathcal{K}(V) = \max_i \sum_j \frac{1}{|b_i - \xi_j|} \max_i \frac{\sigma(\xi_i)}{|S'(b_i)|}$$

A different estimate of the condition number which is specific of a given eigenvalue ξ can be derived from the following classical result.

THEOREM 1.16: *Let A and F be $n \times n$ matrices. If ξ is a simple eigenvalue of A such that $Ax = \xi x$, $y^T A = \xi y^T$, then there exists a neighborhood $\mathcal{U}$ of ξ and an analytic function $\xi(\epsilon) : \mathcal{U} \to \mathbb{C}$ such that*

$$\xi(\epsilon) \doteq \xi + \epsilon \frac{y^T F x}{y^T x}.$$

and $\xi(\epsilon)$ is an eigenvalue of $A + \epsilon F$.

Applying the above theorem to $A = M = D - ae^T$, with $F = e\hat{a}^T$, where $\hat{a} = (\hat{a}_i)$, $\hat{a}_i = a_i\delta_i/\delta$, with $\delta = \max_i |\delta_i|$, since $x_i = 1/(\xi - b_i)$, $y_i = a_i/(\xi - b_i)$, one finds that

$$\frac{y^T F x}{y^T x} = \frac{\left(\sum_{i=1}^n \frac{a_i}{\xi - b_i}\right)\left(\sum_{i=1}^n \frac{a_i\delta_i/\delta}{\xi - b_i}\right)}{S'(\xi)} = \frac{\sum_{i=1}^n \frac{a_i\delta_i/\delta}{\xi - b_i}}{S'(\xi)},$$

where the latter inequality holds since $S(\xi) = 0$ so that $\sum_{i=1}^n \frac{a_i}{\xi - b_i} = 1$. This way, for the variation $\xi(\delta)$ induced by the relative perturbation δ_i on the coefficient a_i, one has

$$|\xi - \xi(\delta)| \ \dot{\leqslant}\ \frac{\delta}{|S'(\xi)|}.$$

This bound can be applied to the case where ξ is such that $\mathrm{fl}(S(\xi)) = \eta$ and $|\eta| \leqslant \hat{\epsilon}$ with $|\epsilon| < 1$. Recall that $\mathrm{fl}(S(\xi)) = (\sum_{i=1}^n \frac{a_i(1+\delta_i)}{\xi - b_i} - 1)(1+\delta)$ where $|\delta_i| \leqslant \kappa_n u$ and $|\delta| \leqslant u$. In fact, in this case, ξ is root of the secular equation $\tilde{S}(x) = 0$ with $\tilde{S}(x) = \sum_{i=1}^n \frac{a_i(1+\delta_i+\eta)}{x - b_i} - 1$. The original function $S(x)$ can be viewed as a perturbation of $\tilde{S}(x)$ where $\delta_i = -\delta_i - \eta$. This way, we find that that there exists a root $\xi(\epsilon)$ of $S(x)$ such that

$$|\xi - \xi(\delta)| \ \dot{\leqslant}\ \frac{\delta}{|S'(\xi)|}, \quad \delta = \max_i |\delta_i + \eta|$$

Moreover, if ξ satisfies the stop condition (1.8) then $|\eta| \leqslant \kappa_n \sigma(\xi) u$ so that

$$|\xi - \xi(\delta)| \ \dot{\leqslant}\ \frac{1}{|S'(\xi)|}(1 + \sigma(\xi))\kappa_n u.$$

1.6 COMPUTATION OF THE NEWTON CORRECTION

The main tool on which our algorithm to compute the secular roots relies is the Erlich-Aberth iteration. This method, which will be introduced next, is based on the Newton iteration. We show in this section how the computation of the Newton correction of the polynomial associated with $S(x)$ may be performed implicitly, without computing the coefficients of $P(x)$.

We recall that the polynomial associated with $S(x)$ can be written as

$$P(x) = \prod_{i=1}^n (x - b_i) S(x) \tag{1.16}$$

and that the Newton correction at the point x is defined by

$$N(x) = \frac{P(x)}{P'(x)}.$$

In the following we will systematically use the notation $N(x)$ for the Newton correction, $S(x)$ for the secular equation and $P(x)$ for the associated monic polynomial.

1.6.1 *Formal computation*

Computing the derivative of $P(x)$ written in the form of equation (1.16) we obtain

$$N(x) = \frac{S(x)}{S'(x) + S(x)\sum_{i=1}^{n} \frac{1}{x-b_i}} = \frac{\frac{S(x)}{S'(x)}}{1 + \frac{S(x)}{S'(x)}\sum_{i=1}^{n} \frac{1}{x-b_i}}. \tag{1.17}$$

A quite straightforward algorithm to compute the value of the Newton correction at x is the one reported in Algorithm 3.

Algorithm 3 Evaluation of the Newton correction using the secular representation.

```
1:  procedure EvaluateNewtonCorrectionSecular(a_i, b_i, x)
2:      s ← 0
3:      s' ← 0
4:      b ← 0
5:      for i = 1 : n do
6:          t ← 1/(x − b_i)
7:          b ← b + t
8:          u ← a_i · t
9:          t ← u · t
10:         s ← s + u
11:         s' ← s' + t
12:     end for
13:     s ← s − 1
14:     d ← s/(s·b + s')
15:     return d
16: end procedure
```

A similar algorithm can be given by replacing the sequential summation by the recursive summation presented in Appendix A.

1.6.2 *Error analysis*

Carrying out the error analysis yields three different bounds on the algorithmic error generated by the evaluation of $S(x)$, $S'(x)$ and $\sum_{i=1}^{n} \frac{1}{x-b_i}$, respectively.

More precisely, if u is the machine precision the error bounds are:

- $|\mathrm{fl}(S(x)) - S(x)| \;\dot{\leqslant}\; \mathrm{fl}(S(x))\,u + \kappa_n \sigma(x) u, \quad \sigma(x) = \sum_{i=1}^{n} \left| \frac{a_i}{x-b_i} \right|$

- $\left|\mathrm{fl}\left(S'(x)\right) - S'(x)\right| \;\dot{\leqslant}\; (\kappa_n + 1)\sigma'(x), \quad \sigma'(x) = \sum_{i=1}^{n} \left| \frac{a_i}{(x-b_i)^2} \right|$

- $\left| \mathrm{fl}\left(\sum_{i=1}^{n} \frac{1}{x-b_i} \right) - \sum_{i=1}^{n} \frac{1}{x-b_i} \right| \;\dot{\leqslant}\; \sum_{i=1}^{n} \kappa_n \sigma''(x) u, \quad \sigma''(x) = \sum_{i=1}^{n} \left| \frac{1}{x-b_i} \right|$

Keeping track of these errors can be done while performing Algorithm 3 without much additional computational effort: nearly all the quantities involved in the bounds (except for the modulus computations) are already computed to evaluate $N(x)$.

This allow to give a guaranteed computation of $N(x)$, that will be useful in giving guaranteed root inclusions.

1.6.3 *Computing Newton correction at the poles*

The secular function $S(x)$ cannot be evaluated at b_i because of the singularity present there. On the other hand, the polynomial $P(x)$ is well defined at $x = b_i$ as well as its derivative. If $P'(x) \neq 0$ then also the Newton correction is well defined at $x = b_i$. However, Algorithm 3 is not applicable for this purpose since it requires an evaluation of $S(x)$ and $S'(x)$ at $x = b_i$.

We present here an alternative algorithm for the case which enables us to compute $N(b_i)$ given the values of b_i and the coefficients a_i for $i = 1, \ldots, n$ of the secular function.

Rewrite $P(x)$ in the following form

$$P(x) = -\sum_{k=1}^{n} a_k \prod_{j=1,\, j\neq k}^{n} (x - b_j) + \prod_{j=1}^{n} (x - b_j).$$

Set $x = b_i$ and get

$$P(b_i) = -\sum_{k=1}^{n} \left(a_k \prod_{\substack{j=1 \\ j\neq k}}^{n} (b_i - b_j) \right) + \prod_{j=1}^{n} (b_i - b_j) = -a_i \prod_{\substack{j=1 \\ j\neq i}}^{n} (b_i - b_j). \tag{1.18}$$

Similarly, for $P'(b_i)$ one obtains

$$P'(b_i) = -\frac{d}{dx} \left(\sum_{k=1}^{n} a_k \prod_{\substack{j=1 \\ j\neq k}}^{n} (x - b_j) \right) \Bigg|_{x=b_i} + \frac{d}{dx} \left(\prod_{j=1}^{n} (x - b_j) \right) \Bigg|_{x=b_i} \tag{1.19}$$

$$= -\sum_{k=1}^{n} a_k \left(\sum_{\substack{l=1 \\ l\neq k}}^{n} \prod_{\substack{j=1 \\ j\neq l,k}}^{n} (b_i - b_j) \right) + \prod_{j=1, j\neq i}^{n} (b_i - b_j) = \prod_{\substack{j=1 \\ j\neq i}}^{n} (b_i - b_j) \left(1 - \sum_{k\neq i} \frac{a_k + a_i}{b_i - b_k} \right). \tag{1.20}$$

Writing the Newton correction using these two expressions we obtain

$$N(x) = \frac{a_i}{\sum_{k\neq i} \frac{a_k + a_i}{b_i - b_k} - 1}.$$

From this last expression we can define another algorithm to evaluate the Newton correction at $x = b_i$ for every $i = 1, \ldots, n$. This procedure is reported as Algorithm 4.

As usual, we perform the error analysis of this procedure. At each step we have that

$$\mathrm{fl}(t) = \frac{a_k + a_i}{b_i - b_k}(1 + 2\epsilon_+ + \epsilon_\div).$$

If we perform a sequential or a recursive sum of all these terms we obtain, at the end of the algorithm,

$$\mathrm{fl}(s) \doteq s(1 + (k_n + 1)u)$$

1.7 SECULAR ROOTS INCLUSIONS

In this section we present some results regarding secular roots inclusions. Our goal is to compute a set of *centers* x_i and *radii* ρ_i such that the union of the discs $B(x_i, r_i)$ of center x_i and radii r_i, $i = 1, \ldots, n$ contains all the roots.

Algorithm 4 Evaluation of the Newton correction when $x = b_i$

```
1: procedure EvaluateNewtonCorrectionSecular(a_i, b_i, i)
2:     /* i is the index such that x = b_i */
3:     s ← 0
4:     b ← 0
5:     for k = 1 : n do
6:         if k ≠ i then
7:             t ← a_k + a_i
8:             b ← b_i − b_k
9:             s ← s + t/b
10:        end if
11:    end for
12:    s ← s − 1
13:    s ← a_i/s
14:    return s
15: end procedure
```

Moreover, we are able to give some results on the number of roots contained in each disc.

1.7.1 *Gerschgorin based results*

Some basic results can be obtained by using Gerschgorin theorems.

THEOREM 1.17 (Gerschgorin): *Let $A \in \mathbb{C}^{n\times n}$ a complex matrix, and consider the set*

$$\Gamma = \bigcup_{i=1}^{n} B(a_{ii}, r_i) \subseteq \mathbb{C}, \quad r_i = \sum_{j\neq i} |a_j| .$$

Then Γ contains all the eigenvalues of A and, in particular, each connected component of it contains exactly a number of eigenvalues equal to the number of circles contained in it. Moreover, if λ is an eigenvalue of A corresponding to the eigenvector x, i.e., $Ax = \lambda x$, then $\lambda \in B(a_k, r_k)$ where k is such that $|x_k| = \max_i |x_i|$.

We will often refer, in the following, to the sets $B(a_{ii}, \rho_i)$ as *Gerschgorin's discs*.

Recalling that the roots of the secular equation are the eigenvalues of the matrix $D - ea^T$ we can use Theorem 1.17 and obtain a bound on the secular roots with circles centered in the nodes b_i.

Considering that

$$D - ea^T = \begin{bmatrix} b_1 - a_1 & -a_2 & \cdots & -a_n \\ -a_1 & b_2 - a_2 & \cdots & -a_n \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ -a_1 & -a_2 & \cdots & b_n - a_n \end{bmatrix}$$

we obtain the following bounds for the eigenvalues ξ_i:

$$\xi_i \in \bigcup_{j=1}^{n} B(b_j - a_j, \sum_{\substack{k=1 \\ k \neq j}}^{n} |a_k|) \subseteq \bigcup_{j=1}^{n} B(b_j, \sum_{k=1}^{n} |a_k|) \tag{1.21}$$

or, considering the transposed matrix,

$$\xi_i \in \bigcup_{j=1}^{n} B(b_j - a_j, (n-1)\left|a_j\right|) \subseteq \bigcup_{j=1}^{n} B(b_j, n\left|a_j\right|). \tag{1.22}$$

Remark 1.18: The result obtained by Gerschgorin's theorem is even more general. If one of the unions in (1.21) or in (1.22) is not connected than we get a partitioning of the roots in the various connected components of the inclusion set.

This will be very useful in the following to detect clusterization properties.

Now recall that an eigenvector x of $A = D - ea^T$ corresponding to the eigenvalue λ is such that $x_i = 1/(x - b_i)$, while if $y^T A = \lambda y$ then $y_i = a_i/(\lambda - b_i)$. Therefore, according to the Gerschgorin theorem, we find the bounds

$$|\lambda - b_k| \leqslant \begin{cases} (n-1)\,|a_i| & \text{if } \left|\frac{a_k}{\lambda - b_k}\right| = \max_i \left|\frac{a_i}{\lambda - b_i}\right| \\ \sum_{i=1,\, i\neq k}^{n} |a_i| & \text{if } |\lambda - b_k| = \max_i |\lambda - b_i| \end{cases}$$

1.7.2 *Gerschgorin bounds and root neighborhood*

The result shown in Section 1.7.1 can be used to give a posteriori bounds on the approximation errors of the computed roots. Since we desire to give *guaranteed* bounds we need to be certain that the bound computed in floating point is greater than or equal to the actual one.

As usual, this involves performing a first order error analysis of the algorithms for computing the bounds.

Let us start by considering the bound based on the discs of radius $n \cdot |a_i|$. The error on the radius evaluation is bounded by $n \cdot \epsilon_i$ where ϵ_i is the error in the computation of the coefficient a_i .

Referring to Section 1.4.1 we obtain that $\mathrm{fl}(a_i) = a_i(1+\epsilon_i)$ where $\epsilon_i \leqslant (\kappa_n + 1)u$, and so

$$|a_i| \leqslant |\mathrm{fl}(a_i)|\,(1+\epsilon_i).$$

This last consideration means that the valid floating point inclusion is

$$\xi_i \in \bigcup_{j=1}^{n} B(b_j, n\left|\mathrm{fl}\left(a_j \cdot (1+\epsilon)\right)\right|),$$

that can be seen as the Gerschgorin inclusion of all the roots of the secular equations contained in $\mathrm{RN}_\epsilon(S)$, i.e. the union of the inclusion discs for all secular equation whose coefficients a_i are modified with a relative perturbation smaller than ϵ.

2

SIMULTANEOUS APPROXIMATION

2.1 POLYNOMIAL EVALUATION

In order to approximate the roots of the polynomials and to obtain precise roots bounds we must be able to evaluate a polynomial (and its derivatives) at a point with high precision and efficiency.

The main tool used to achieve this in the case where the polynomial is given in terms of its monomial coefficients, is the Horner scheme.

2.2 THE HORNER SCHEME

2.2.1 *Basic Horner scheme*

The Horner scheme is an algorithm used to evaluate a polynomial $P(x) = \sum_{i=0}^{n} p_i x^i$ at a given point $\bar{x}$.

The basic Horner algorithm can be described by the following expression:

$$P(\bar{x}) = (((p_n \bar{x} + p_{n-1})\,\bar{x} + p_{n-2})\,\bar{x} + \ldots + p_1)\,\bar{x} + p_0.$$

The pseudocode is described in Algorithm 5.

Algorithm 5 Horner algorithm

```
1: procedure HORNER(P, x̄)
2:     v ← a_n · x̄
3:     for i = n − 1, ..., 0 do
4:         v ← v · x̄ + p_i
5:     end for
6:     return v
7: end procedure
```

The big advantages of the Horner scheme on other naive implementations of the polynomial evaluation are the *low computational cost* (that is n multiplication and $n-$

1 additions) and the property of *backward stability*. Moreover, it can be adjusted to compute the derivatives of the polynomial as well.

2.2.2 *Computing the derivatives of* $P(x)$

A slight modification of the Horner algorithm can be used to compute the derivatives, and consequently the Taylor expansion, of the polynomial P at the point $\bar{x}$.

Let v_k be the value of v in Algorithm 5 when $i = n - k$, so that $v_0 = a_n\bar{x}$, $v_1 = v_0\bar{x} + p_{n-1}$ and so on. Clearly we have that $v_n = P(\bar{x})$.

If we put $b_{i,-1} = p_i$ we can perform this Horner-like iteration with two indices, valid for $-1 \leqslant j \leqslant i \leqslant n$:

$$\begin{cases} v_{0,1} & = v_{0,0} \\ v_{i,j} & = v_{i,j-1} + \bar{x}v_{i-1,j} \end{cases}. \tag{2.1}$$

As shown in [Hen88] this sequence has the property that $v_{n-i,i} = P^{(i)}(\bar{x})$, and this allow us to write the Taylor expansion of the polynomial P. Note that this is particularly helpful when our goal is to compute a small number of the coefficients of the shifted polynomial $P(x - \bar{x})$.

Observe that the iteration (2.1), applied with $j = 0, 1$, provides the Newton correction $\frac{P(\bar{x})}{P'(\bar{x})}$.

2.3 THE DURAND-KERNER METHOD

The Durand-Kerner method, also known as Weierstrass method, is one of the first techniques introduced for the simultaneous approximation of polynomial roots.

2.3.1 *The iteration*

The method is described by the following iteration

$$x_i^{(k+1)} = x_i^{(k)} - \frac{P(x_i^{(k)})}{\prod_{j \neq i}(x_i^{(k)} - x_j^{(k)})} \tag{2.2}$$

where $x_i^{(k)}$ and $x_i^{(k+1)}$, for $i = 1, \ldots, n$, are the approximations to the roots of $P(x)$ at two subsequent steps of the iteration.

An interesting derivation of this formula has been given by Aberth in [Abe73]. One may try to request that $x_i^{(k+1)} = \xi_i$ where $\xi_1, \ldots, \xi_n$ are the roots of P. Let $\Delta_i = x_i^{(k+1)} - x_i^{(k)}$, so that if $\xi_i = x_i^{(k+1)}$ the following relation holds:

$$P(x) = \prod_{i=1}^{n}(x - \xi_i) = \prod_{i=1}^{n}(x - x_i^{(k)} + \Delta_i).$$

If we keep only the linear terms in Δ_i in the above expression we have

$$P(x) \doteq \prod_{i=1}^{n}(x - x_i^{(k)}) - \sum_{i=1}^{n} \Delta_i \prod_{j \neq i}(x - x_i^{(k)}).$$

Setting $z = z_i^{(k)}$ and solving by Δ_i we find iteration (2.2).

2.3.2 *Quadratic convergence*

As shown for example in [McNo7] this method has local quadratic convergence. Moreover, even if convergence turns to linear in the case of multiple roots, Fraigniaud has shown in [Fra91] that quadratic convergence is still maintained by the mean of the approximations convergent to a multiple root.

2.4 THE ERLICH-ABERTH METHOD

The Erlich-Aberth method (introduced independently by Aberth [Abe73], Börsch-Soupan [BS63] and Ehrlich [Ehr67]) is a numerical engine that aims to approximate all the roots of a polynomial simultaneously. It is based on the Newton method and the idea of implicit deflation.

2.4.1 *Implicit deflation*

We can derive the iteration of the Erlich-Aberth method trying to apply the Newton method to a (hopefully) good approximation of the deflated polynomial.

More precisely if we want to approximate the k-th root of the polynomial assuming that the other roots are known to be

$$\zeta_0, \ldots, \zeta_{k-1}, \zeta_{k+1}, \ldots, \zeta_n,$$

we can apply the Newton method to the linear function

$$f_k(x) = \frac{P(x)}{\prod_{i \neq k}(x - \zeta_i)}$$

and obtain convergence in one step. Modifying this simple observation to suit the case of simultaneous approximation, we can apply the Newton method to the rational function

$$g_k(x) = \frac{P(x)}{\prod_{i \neq k}(x - \xi_i)}$$

where ξ_i are the approximation obtained at the previous step. Obviously g_k is not a linear function as in the case analyzed before, but if the approximation are near enough to the roots of P then g_k will become a good approximation of f_k, at least in a proper neighborhood of ξ_k.

2.4.2 *Convergence*

Erlich-Aberth method has local cubic convergence in every simple root (as shown in [Abe73]). In the case of multiple roots of multiplicity m, the convergence rate is only linear with an error reduction of $O(1 - \frac{1}{m})$.

This fact can be seen by simply expanding the iteration with its Taylor series centered at the zero, and observing that derivatives are zero until the third order.

REMARK 2.1: Observe that in order to apply Aberth's iteration it is sufficient to have a "black box" which provides the ratio $\frac{P(x)}{P'(x)}$ for a given input value x. This allows to apply the method to polynomials represented in different basis, not necessarily the monomial one.

2.5 THE BAIRSTOW METHOD

In this section we recall the Bairstow method for computing real quadratic factors of a real polynomial, together with a more recent version which implements a simultaneous iteration technique.

2.5.1 *Classic Bairstow method*

The classic Bairstow method, first introduced by Bairstow in [Bai20], has the purpose of approximating a pair of complex conjugated roots of a real polynomial.

The main advantages of this method are the following:

1. Only real arithmetic is needed during computation; this can give a good increase in the performance and much easier handling of overflow and underflow conditions;

2. The structure of the problem is used and maintained, so the roots computed by the algorithm are guaranteed to be real or complex conjugated pairs;

The method approximates the coefficients of a quadratic factor of the polynomial associated with a pair of complex conjugated roots.

More precisely, suppose that we want to approximate the pair of roots $\{\zeta_k, \overline{\zeta}_k\}$ of the polynomial $P(x)$, or equivalently the quadratic factor $x^2 + rx + q = (x - \zeta_k)(x - \overline{\zeta}_k)$ that divides $P(x)$.

Let r_0, q_0 be an initial guess for the coefficients of the quadratic factor. We can compute the euclidean division of $P(x)$ by the polynomial $x^2 + r_0 x + q_0$ that yields:

$$P(x) = (x^2 + r_0 x + q_0)Q(x) + cx + d.$$

Clearly, the quadratic factor divides $P(x)$ if and only if the vector $c = d = 0$.

The coefficients of the remainder can be seen as function of r, q and we may apply 2-dimensional Newton iterations to approximate the coefficients r, q.

It can be easily seen that if we divide $Q(x)$ by $x^2 + rx + q$, obtaining the remainder $x^2 + c_1 x + d_1$ we can write the Jacobian as a function of r, q, c_1, d_1 and obtain the following Newton iteration:

$$\begin{bmatrix} r_{k+1} \\ q_{k+1} \end{bmatrix} = \begin{bmatrix} r_k \\ q_k \end{bmatrix} - \begin{bmatrix} c_1 r_k + d_1 & c_1 \\ c_1 q_k & d_1 \end{bmatrix}^{-1} \begin{bmatrix} c \\ d \end{bmatrix}.$$

The values c, d, c_1 and d_1 can be evaluated at every step with a Horner-type scheme.

2.5.2 *The parallel implementation*

To implement the Bairstow idea following an "Aberth-like philosophy" we need a way of implicitly deflating the polynomial by the approximated factor $x^2 + p_k x + q_k$.

A possible way to achieve this is exposed by Luk in [Luk96] referring to an older procedure developed by Handscomb in [Han62].

The classical Bairstow method proceeds computing $R(x) = P(x) \mod x^2 + rx + q$ at every step and trying to find r, q such that $R(x) \equiv 0$. The simultaneous version of the algorithm, instead, performs an implicit deflation using the approximation of the other quadratic factors. So, to approximate the k-th pair of roots the Newton method is applied to:

$$R_k(x) = \left(\frac{P(x)}{\prod_{j \neq k}(x^2 + r_j x + q_j)} \right) \mod (x^2 + r_k x + q_k).$$

Remark 2.2: The Bairstow method here exposed can be seen as a generalization of the Aberth method. If we replace the approximation of the quadratic factors with linear factors over $\mathbb{C}[x]$ we obtain exactly that

$$\frac{P(x)}{\prod_{j \neq k}(x - \xi_j)} \mod (x - \xi_j) = \frac{P(\xi_j)}{\prod_{j \neq k}(x - \xi_k)}.$$

2.5.3 *Cost of the Bairstow iteration*

The Bairstow iteration exposed here is more expensive in terms of arithmetic operation than the classic Aberth iteration. Nevertheless, the fact that only real arithmetic is necessary can give a gain in performance which compensates that, so the method can still be quite useful in practice.

Moreover, the approximated roots will be guaranteed to occur in complex conjugate pairs.

3

POLYNOMIAL'S ROOTS INCLUSIONS

The problem of approximating polynomial roots is connected to the one of giving precise and guaranteed bounds to the error affecting these approximations.

We will present some theorems and results that allow to state different type of inclusion for polynomial roots, typically in discs with centers in the approximations computed at the general step.

A detailed analysis of these and many other inclusion results can be found in [Hen88].

3.1 NEWTON INCLUSIONS

3.1.1 *Preliminary results*

We are interested in a set of inclusions related to the Newton correction of the polynomial computed at a point $\bar{z}$. These results are particularly useful since both Aberth method and Newton method must compute the value of $\frac{P(\bar{z})}{P'(\bar{z})}$ and so inclusions based on this ratio are often "free" in terms of computational cost.

We have already seen in Section 2.1 how to compute the coefficients of the Taylor expansion of P at a point $\bar{x}$.

THEOREM 3.1: *Let w_i be the i-th coefficient of the Taylor expansion of the polynomial $P(x)$ in $\bar{x}$, i.e. the i-th coefficient of the shifted polynomial $P(x-\bar{x})$. If we define*

$$\beta(x) = \min_{1 \leqslant m \leqslant n} \left[\binom{n}{m} \left| \frac{w_0}{w_m} \right| \right]^{1/m}$$

then $B(\bar{x}, \beta(\bar{x}))$ contains at least one root of $P(x)$.

This theorem induces a family of corollaries simply by changing the value of m. For instance, for $m = n$ and $m = 1$ we obtain the following corollaries.

COROLLARY 3.2: *Let $P(x)$ be a polynomial of degree n. There is always at least a root of it contained in the set $B(\bar{x}, P(\bar{x})^{\frac{1}{n}})$.*

COROLLARY 3.3: *In the same hypothesis of Corollary 3.2 we have that the set* $B(\bar{x}, n\frac{P(\bar{x})}{P'(\bar{x})})$ *always contains a root of the polynomial.*

Those two corollaries are obtained by putting respectively $m = n$ and $m = 1$ in Theorem 3.1.

3.1.2 *Higher order inclusion*

In some cases it may be interesting to find inclusions results for k roots, instead of only one. This is particularly helpful, for example, if we are trying to separate a cluster of roots from the others.

Observe that if the coefficients of the Taylor expansion at $\bar{x}$ are such that

$$w_0 = w_1 = \ldots = w_m = 0$$

then P has a zero of multiplicity (at least) m in $\bar{x}$.

This suggests that, by continuity of the roots with respect to the coefficients, if those coefficients are small enough then at least m roots of P are close to $\bar{x}$.

We would like to give a formal statement of this vague intuition, reporting a result due to Montel.

THEOREM 3.4: *Consider the following polynomial in the variable* ρ*:*

$$\rho^n - \binom{n-1}{m-1}|w_0| - \binom{n-2}{m-2}|w_1|\,\rho - \ldots - \binom{n-m}{0}|w_m|\,\rho^{m-1}.$$

This polynomial has always a unique non negative root ρ_m *and the disc* $B(\bar{x}, \rho_m)$ *contains at least* m *roots of the polynomial* P*.*

In many situations we have not only that $w_0, \ldots, w_m$ are small, but also that w_{m+1} is much bigger than w_m. That will be the typical case in which we identify a cluster of m roots at $\bar{x}$.

It may be advantageous, in these cases, to use the following result due to Van Vleck.

THEOREM 3.5: *Let* $w_m \neq 0$*. Consider the following polynomial in the variable* ρ*:*

$$|w_m|\,\rho^n - \binom{n-1}{m-1}|w_0| - \binom{n-2}{m-2}|w_1|\,\rho - \ldots - \binom{n-m}{0}|w_m|\,\rho^{m-1}.$$

This polynomial has always a unique non negative root ρ^*_m *and the disc* $B(\bar{x}, \rho^*_m)$ *contains at least* m *roots of the polynomial* P.

3.2 GERSCHGORIN INCLUSIONS

3.2.1 *From polynomials to linear algebra*

In this section we analyze Gerschgorin-type inclusion, i.e. inclusions based on Gerschgorin's theorems. These results allow to obtain some discs that are not guaranteed to contain a root, but such that every connected component of their union contains exactly m roots where m is the number of discs which form the component.

We use Theorem 1.17 and perform an analysis almost identical to the one in Section 1.7.

Suppose that we are given a polynomial $P(x)$ and a set of approximations x_i. We may construct a companion-like matrix $C(P, x)$ as in Definition1.2:

$$C(P,x) = \begin{bmatrix} x_1 & & \\ & \ddots & \\ & & x_n \end{bmatrix} - \begin{bmatrix} a_1 & \cdots & a_n \\ \vdots & & \vdots \\ a_1 & \cdots & a_n \end{bmatrix} \qquad \text{where } a_i = \frac{P(x_i)}{p_n \prod_{\substack{j=1 \\ j\neq i}}^{n} (x_i - x_j)}.$$

We have already seen that the eigenvalues of this matrix are exactly the roots of P and so, by simply applying Gerschgorin theorem, we obtain a set of bounds.

According to Gerschgorin theorem the roots $\xi_1, \ldots, \xi_n$ are contained in the union of the discs of center x_i and radius $n\,|a_i|$. Moreover, if this union is made up of several connected components, every one of these contains a number of roots equal to the number of discs that form the component.

3.2.2 *Guaranteed radius computation*

An immediate problem that affects all these results on root inclusions is that we work in floating point arithmetic and not really on the complex field. Nevertheless, we would like to have *guaranteed* inclusions, i.e., we would like to bound the error that affects the computation and give certainly true statements about root's positions.

3.3 INCLUSION RESULTS FOR SELECTING STARTING POINTS

We present a method, used in `MPSolve`, that relies on the knowledge of the polynomial coefficients in the monomial basis to determine some inclusion discs. This method has proved to be particularly effective to distinguish "small" and "large" roots.

The results that we present here give inclusion results independent of the approximation obtained at a given step of the iteration. For this reason these are particularly effective to find good starting points for the approximation algorithms.

In the case of secular equations, instead, a simpler method will be used based on the observations made in Section 1.7.

We see how it is possible to combine those two strategies to obtain a composite polynomial approximation algorithm that exploits the advantages of both strategies.

3.3.1 *Choice of starting points based on the Rouché theorem*

The first algorithm for choosing the starting points that we inspect here is the one presented in [Bin96], based on the Rouché theorem.

THEOREM 3.6 (Rouché): *Let* f, g *be two holomorphic functions defined on a simply connected domain* D *with boundary* Γ, *and suppose that the following relation holds*

$$|f(x)| > |g(x)| \qquad \textit{for every } x \in \Gamma.$$

Then the number of zeros of f *in* D *is equal the the number of zeros of* $f + g$ *in* D.

Suppose that we are given the polynomial

$$P(x) = \sum_{i=0}^{n} p_i x^i.$$

Choose k such that $p_k \neq 0$. We may then consider $f(x) = p_k x^k$ and $g(x) = P(x) - p_k x^k$. If we find a positive real number r such that

$$\left|p_k x^k\right| > \left|P(x) - p_k x^k\right| \qquad \text{for every } x,\ |x| = r,$$

then we may apply Theorem 3.6 and obtain that P has exactly k roots in $B(0, r)$. Instead of trying to find this r directly we may observe that

$$\left|P(x) - p_k x^k\right| \leqslant \sum_{\substack{i=0 \\ i \neq k}}^{n} |p_i| |x|^i ,$$

where equality is reached for some values of the coefficients with the same modulus of $|P_i|$. So if we find a positive real θ that solves

$$|p_k| \theta^k - \sum_{\substack{i=0 \\ i \neq k}}^{n} |p_i| \theta^i = 0 \tag{3.1}$$

then $\left|p_k x^k\right| \geqslant \left|P(x) - p_k x^k\right|$ for $|x| = \theta$.

It's possible to show that, for every $0 < k < n$, this equation can have two or zero positive solutions, while in the case of $k = 0, n$ there exists exactly one solution.

We call these solutions $t_k < s_k$ whenever they exist, and we call s_0 the positive solution for $k = 0$ and t_n the one for $k = n$.

THEOREM 3.7: *Let θ be a positive solution of equation (3.1) for a given k. Then the closed disc $\overline{B(0, \theta)}$ contains k roots of P.*

Proof. It follows directly from the previous statements. □

Consider now a solution θ such that

$$|p_k| \theta^k > \sum_{\substack{i=0 \\ i \neq k}}^{n} |p_i| \theta^i .$$

We have that, in particular, $|p_k| \theta^k > |p_i| \theta^i$ for every $i \neq k$. So we can write, for every $i < k < j$

$$\left|\frac{p_i}{p_k}\right|^{\frac{1}{k-i}} < \theta < \left|\frac{p_j}{p_k}\right|^{\frac{1}{k-j}} .$$

The inequality will be still true if we take the maximum of the left member for $i < k$ and the minimum of the right one for $j > k$. We refer to these two values respectively as u_k and v_k.

Clearly, if a solution θ do exists, we have that $u_k < v_k$.

THEOREM 3.8: *Let* $0 = k_1 < k_2 < \ldots < k_{q-1} < k_q = n$ *the values of* k *such that equation* (3.1) *has at least one positive solution. Let* $t_0 = 0$ *and* $s_n = +\infty$. *Then, the polynomial* P *has* $k_{i+1} - k_i$ *solutions in every annulus*

$$\mathcal{A}_i = \left\{ z \in \mathbb{C} \,|\, s_{k_i} \leqslant |z| \leqslant t_{k_{i+1}} \right\}.$$

and the annuli $\{t_{k_i} \leqslant |z| \leqslant s_{k_i}\}$ *do not contain roots of* P.

These theorems, with some auxiliary propositions, are all proved in [Bin96], where we refer for a deeper analysis.

These observations suggest a method for choosing the starting points. We may choose $k_{i+1} - k_i$ starting points equally distributed on a circle of radius $\frac{s_{k_i} + t_{k_{i+1}}}{2}$. The problems in this approach are determining the values of k such that equation (3.1) has positive solutions, and then computing the values of s_{k_i} and t_{k_i} explicitly.

A possible method to overcome this issues is to determine the values of k such that $u_k \leqslant v_k$. This is a necessary condition in order that the equation $|p_k| x^k = \sum_{i=0 i \neq k}^{n} |p_i| x^i$ ha real positive solution. Suppose that $k_1, \ldots, k_n$ are those indices; we will the use $k_{i+1} - k_i$ equally distributed points on the circle of radius $u_{k_{i+1}}$ as starting points. As clarified by the next paragraph, these points will be included in the annulus $\{t_{k_i} \leqslant |x| \leqslant s_{k_{i+1}}\}$.

3.3.2 *The Newton polygon*

Here we present a technique used to compute the u_i, based on the Newton polygon. Consider the set of points of coordinates $(i, \log(|p_i|))$ for i from 0 to n, $p_i \neq 0$. In the case where where $p_i = 0$ we may consider the point $(i, -\infty)$ or simply ignore these coefficients.

We are interested in computing the upper part of the convex hull of these points. This will be identifiable with a subset $\{k_i \,|\, i = 1, \ldots, q\}$ of the indices $1, \ldots, n$. The convexity condition implies that the slopes of the linear function connecting $(k_i, \log(|p_{k_i}|))$ and $(k_{i+1}, \log(|p_{k_{i+1}}|))$ will be decreasing.

These slopes are given by

$$\alpha_i = \frac{\log(|p_{k_{i+1}}|) - \log(|p_{k_i}|)}{k_{i+1} - k_i}.$$

Setting $\log(r_i) = \alpha_i$ we have that $r_i = \left|\frac{a_{i+1}}{a_i}\right|^{\frac{1}{k_{i+1}-k_i}}$, that are the exponential of the slopes.

In [Bin96] is shown that $r_i = u_{i+1} = v_i$ and so using r_i as radii for the initial collocation of the approximations yields a choice of starting points contained in the annuli given by Theorem 3.8.

The convex hull of a set of points in the plane can be computed in $O(n)$, using the Graham scan exposed in [Gra72].

3.3.3 *Maximizing starting points distances*

It has been empirically observed that when placing the initial approximations it is preferable not to choose starting points close to each other. In fact, Aberth's method may encounter convergence problems if started with initial approximations that are too near.

This is the reason that suggested to choose the starting approximations equally spaced on circles of the radii obtained by the Newton polygon.

Even using this technique, there may be cases where two radii are relatively close and this could cause two starting points to be quite near even on different circles.

There are two possible approaches to solve this issues, and both will be simultaneously applied in the final algorithm.

The first idea is to collapse circles that are too near.

Let r_1 and r_2 be the two radii determined by the Newton polygon. Suppose that the approximations chosen are

$$\left\{r_1 e^{\frac{2i\pi}{m_1}} \in \mathbb{C}\,,\ i = 1,\dots,m_1\right\} \cup \left\{r_2 e^{\frac{2j\pi}{m_2}+\sigma}\,,\ j = 1,\dots,m_2\right\}.$$

We would like to decide if the strategy of using a single circle of radius $\tilde{r} = \frac{m_2 r_1 + m_1 r_2}{m_1+m_2}$, i.e. of choosing the initial approximations

$$\left\{\tilde{r} e^{\frac{2i\pi}{m_1+m_2}}\,,\ i = 1,\dots,m_1+m_2\right\}$$

is convenient or not.

For this purpose, we compute the minimum of the distance between the starting approximation in both cases, and select the strategy that maximize this minimum.

In the second case we are looking for a shift of the angle of the placement on the circles, selecting a sequence of angles σ_i for $i = 1,\dots,t-1$ such that the distance from

the points is maximum when rotating the placement on the $(i+1)$-th circle by σ_i (and not rotating the first). We need to solve the minimax problem associated with this issue.

3.3.4 *Solving the starting minimax problem*

Suppose to have a set of initial radii r_i, such that $r_1 < \ldots < r_t$, with m_i roots to place on every circle. We want to determine $\sigma_1, \ldots, \sigma_{n-1}$ that maximize the minimum of the distances of the starting approximations, i.e. such that

$$\min_{(i,k)\neq(j,l)} \left\| r_i e^{\frac{2k\pi}{m_i}} - r_j e^{\frac{2l\pi}{m_j}} \right\|_2$$

is maximized for every $1 \leqslant i, j \leqslant t$, $1 \leqslant k \leqslant m_i$ and $1 \leqslant l \leqslant m_j$.

Consider, for now, only two circles of radius $r_1 < r_2$, and suppose that the bigger circle is rotated of an angle σ.

The distance between two points can be written as an increasing monotone function of the angle α for $0 \leqslant \alpha \leqslant \pi$, and so we may minimize the difference of the angle between the directions of these points.

The difference α_{ij} is

$$\alpha_{ij} = 2\pi \left(\frac{i}{m_1} - \frac{j}{m_2} \right) - \sigma.$$

Our objective is choosing σ^* in a way that maximizes $\min_{i,j} \alpha_{ij}$.

THEOREM 3.9: *One of the values of* σ *that realizes the maximum of* $\min_{ij} \alpha_{ij}$ *is*

$$\sigma^* = \frac{\pi}{\operatorname{lcm}(m_1, m_2)}$$

where $\operatorname{lcm}(m_1, m_2)$ *is the least common multiple of* m_1 *and* m_2.

Proof. We may suppose without loss of generality that i and j are in $\mathbb{Z}$, since the minimum of the distances does not change. We may then write

$$\min_{i,j} \left(\frac{i}{m_1} - \frac{j}{m_2} \right) = \frac{1}{m_1 m_2} \min_{i,j} (m_2 i + m_1 j) = \frac{\operatorname{lcd}(m_1, m_2)}{m_1 m_2} = \frac{1}{\operatorname{lcm}(m_1, m_2)}.$$

Using Bezout's identity we have that the application

$$\begin{array}{rccc} \phi_{m_1,m_2}(i,j): & \{1,\dots,m_1\}\times\{1,\dots,m_2\} & \to & \mathbb{R} \\ & (i,j) & \longmapsto & 2\pi\left(\frac{i}{m_1}-\frac{j}{m_2}\right) \end{array}$$

has the set

$$\left\{2\pi\frac{k}{\operatorname{lcm}(m_1,m_2)} \mid k=0,\dots,\operatorname{lcd}(m_1,m_2)\right\}$$

as image. This implies that any value of α of the form $\frac{2\pi(k+0.5)}{\operatorname{lcm}(m_1,m_2)}$ maximizes the distance of σ from the image. Setting $k=0$ we obtain

$$\sigma^* = \frac{\pi}{\operatorname{lcm}(m_1,m_2)}.$$

□

This result suggests a method for finding a possibly good approximation of the solution of the placement problem. We may start from placing the points on the first circle with $\sigma=0$, and then proceed on the others in increasing order determining the shift σ so that we reach the maximum distance between every circle and the previous one.

REMARK 3.10: Note that this solution is not automatically the solution of the global minimax problem involving all the circles. But if we combine this strategy with the one of collapsing the circles with close radii we have good chances that the circles are enough separated in a way that the minimum of the distance is realized on the same circle or on two subsequent ones. If that is the case, our solution is the real solution of the minimax.

3.3.5 *Finding starting points for secular equations*

We'll now analyze the problem of determinate suitable starting points for the approximation of the roots of a secular equation

$$S(x) = \sum_{i=0}^{n} \frac{a_i}{x-b_i} - 1.$$

We have seen in Section 1.7 that b_i and a_i may be used to give Gerschgorin-like inclusions for the roots. This means that at the starting of the algorithm, with basically no computation at all, we already have a set of results on the position of the roots.

More precisely, we know that they are contained in the union of the circles $B(b_i, n|a_i|)$, and so it seems natural to choose precisely the b_i as starting approximations. One may observe that since the b_i are poles of the secular function it's not possible to evaluate $S(x)$ if this choice of starting points is made. This is not a problem since we have already described in Algorithm 4 how it is possible to compute the Newton correction of the associated polynomial even at the points b_i.

The use of the Newton polygon to detect the starting approximations may seem a fallback choice used mainly because is cheap and we are not able to find s_k and t_k easily. Actually section 3.4 will highlight how this strategy is very appropriate to find sharp bounds, and will even refine some of the results that are reported here.

3.4 INCLUSION RESULTS BASED ON TROPICAL ALGEBRA

In this section we show some other results regarding polynomial roots inclusions. More precisely, we will show a connection between a polynomial in $\mathbb{K}[x]$ where $\mathbb{K}$ is $\mathbb{R}$ or $\mathbb{C}$ and the polynomials in a tropical semiring.

We then show that the knowledge of the *tropical roots* of this polynomial give us some inclusion results on the root of the original polynomial.

Here, we highlight some of the results obtained by Sharify in [Sha11] and by Sharify and Gaubert in [GS09].

3.4.1 *General notions of tropical algebra*

DEFINITION 3.11: The following operations defined on $\mathbb{R}^+$

$$\begin{cases} a \oplus b := \min(a, b) \\ a \otimes b := a + b \end{cases}$$

are such that $(\mathbb{R}^+, \oplus, \otimes)$ has a semiring structure, and is called the *tropical semiring*.

DEFINITION 3.12: A *tropical polynomial* of degree n is a polynomial on the tropical semiring, i.e. a formal expression of the form:

$$tP(X) = \bigoplus_{k=0}^{n} P_k X^k$$

From the definition that we have given of the operations on the tropical semiring, it can be easily seen that a tropical polynomial is identifiable with a piecewise linear function on $\mathbb{R}^+$; precisely, expanding the definition of $\oplus$ and $\otimes$ we get a function $f(x)$ of the form:

$$f(x) = \max(P_k + k \cdot x).$$

It has been shown in [CGM80] that there exists a unique n-tuple of positive coefficients $c_1, \ldots, c_n$ such that

$$f(x) = P_n + \sum_{i=1}^{n} \max(x, c_k)$$

and these c_k will be called the *tropical roots* of the polynomial $tP(X)$.

3.4.2 *Computing tropical roots*

Consider, as an example, the tropical polynomial $tP(x) = x^3 \oplus 2x^2 \oplus 7x \oplus 8$. If we plot the graph of the associated piecewise linear function we obtain the one shown in Figure 3.1.

This polynomial has two tropical roots with multiplicity 1 and 2.

These roots can be easily computed in $O(n \log n)$ time using the Graham scan (see [Gra72]) applied to the computation of the Newton polygon associated to tP, i.e. the upper convex hull of the points $(i, \log a_i)$.

The opposite of the slopes of the linear piece of the Newton polygon are exactly the tropical roots, as is shown in Figure 3.2 for the polynomial considered in our example.

3.4.3 *Using tropical roots to find classical roots*

Recall that, in the previous section, we have used these roots as a starting radius for the approximations. This paragraphs justifies this choice. In [Sha11] one can find two results (reported here) that give us inclusions of the classical roots of P in specific annuli, that are more strict the more the tropical roots are separated.

Figure 3.1.: Piecewise linear function associated to $tP(x) = x^3 \oplus 0.5x^2 \oplus 7x \oplus 8$

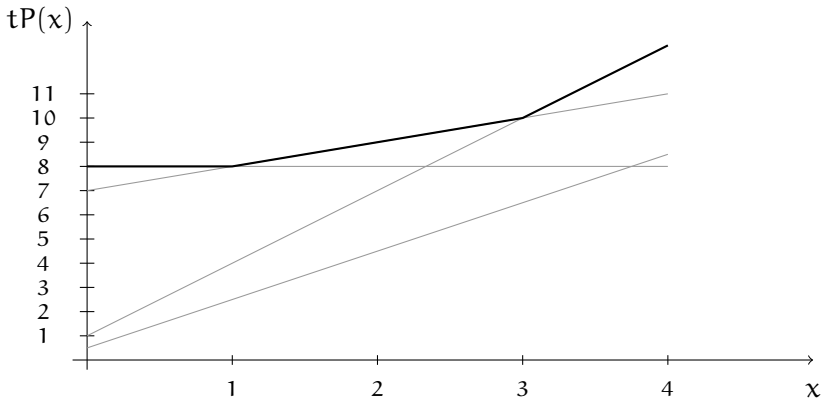
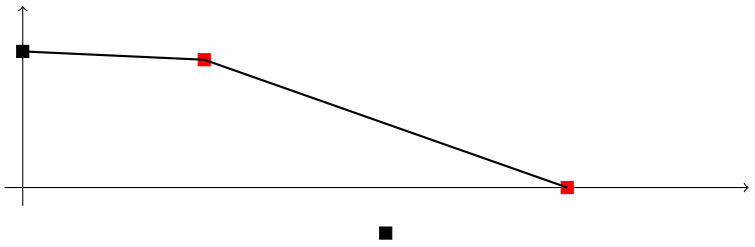


Figure 3.2.: Newton polygon associated to $tP(x) = x^3 \oplus 2x^2 \oplus 7x \oplus 8$. The vertexes corresponding to the tropical roots are highlighted in red.



Given the polynomial $\sum_k a_k x^k$ we can consider the associated tropical polynomial given by $tP(X) = \sum_k |a_k| x^k$ and we have that, if the tropical roots are well separated, it is possible to determine some annuli where the classical roots are contained.

THEOREM 3.13: *Suppose that, for some* $i = 1, \dots, n-1$ *the following conditions holds:*

$$\alpha_i > 9\alpha_{i-1} \text{ and } \alpha_i < \frac{1}{9}\alpha_{i+1}.$$

Then the annulus

$$\left\{ \frac{1}{3}\alpha \leqslant |\zeta| \leqslant 3\alpha_i \right\}$$

contains exactly m_i *roots of* $P(x)$*, where* m_i *is the multiplicity of the tropical root* α_i*.*

THEOREM 3.14: *If the following relation holds for some* i

$$\alpha_i > (2^{m_i+1} + 2)\alpha_{i-1} \text{ and } \alpha_i < \frac{1}{2^{m_i+1} + 2}\alpha_{i+1}.$$

where m_i *is the multiplicity of the tropical root* α_i*, then the annulus*

$$\left\{ \frac{1}{2}\alpha \leqslant |\zeta| \leqslant 2\alpha_i \right\}$$

contains exactly m_i *roots of* $P(x)$*..*

REMARK 3.15: This result clarifies the connection between the Rouché based inclusions presented in Section 3.3 and the use of the Newton polygon to approximate the values s_{k_i} and t_{k_i}. Moreover, with this technique we are able to give precise bounds for the inclusions whenever there is a strict separation of the tropical roots. This justifies the strategy that we have proposed in Section 3.3.3 of collapsing the radii that are too close.

3.5 ROOTS ISOLATION AND CONVERGENCE RATES

3.5.1 *The Newton method*

In this section, we are interested to analyze the behaviour of a polynomial root's approximation when the Newton method is applied to it.

By the basic theory of the Newton method, we know that, if the root is simple, there exists a neighborhood $\mathcal{U}$ such that the sequence $\{x_k\}$ obtained by starting from any point $x_0 \in \mathcal{U}$ is quadratically convergent. In general it can be quite difficult to give results on this neighborhood, a part the fact that it exists.

Some interesting results on this topic are covered in [Til98]. We report here the theorems that will be useful in the development of our algorithm.

THEOREM 3.16: *Let $P(x)$ be a monic polynomial of degree n and $\xi_1, \ldots, \xi_n$ its complex roots. Suppose that $n \geqslant 4$ and the roots are pairwise distinct. If α is a real number such that*

$$|\alpha - \xi_1| \leqslant \frac{1}{3(n-1)} |\alpha - x_i| \quad \textit{for } i = 2, \ldots, n$$

then the sequence generated by the Newton method applied to P with α as starting point is quadratically convergent, and more precisely if we denote with $\alpha^{(m)}$ the m-th element of this sequence we have that

$$\left|\alpha^{(m)} - \xi_1\right| \leqslant \frac{1}{2^{2^m - 1}} |\alpha - \xi_1|$$

REMARK 3.17: In the light of the previous theorem we may note that for the purpose of approximating the roots of a polynomial the real difficult step is finding some approximations that satisfy the hypothesis of the previous theorem. Once this is done, it's not difficult to obtain arbitrary precision approximations simply by applying the pure Newton method and controlling the error with the bound just obtained.

In the following, we will often refer to the first step as *isolation* and to the second step as *refinement* of the roots.

3.5.2 *Aberth's method*

Similar results to the ones of the previous section can be achieved when studying Aberth's method. Aberth's method has a local cubic convergence and so we'll give neighborhoods of the roots such that the convergence is "cubic from the start".

THEOREM 3.18: *Let $P(x)$ be a monic polynomial of degree $n > 9$ with pairwise distinct roots. Let α_i be a set of approximations of the roots $\xi_1, \ldots, \xi_n$. Suppose that for every $i = 1, \ldots, n$ the following relation holds*

$$|\alpha_i - \xi_i| \leqslant \frac{1}{3\sqrt{n-1}} \left|\alpha_i - \alpha_j\right| \quad \textit{for } i \neq j.$$

If we consider the sequence of vectors $\alpha^{(m)}$ *whose components are the points obtained as approximations for the roots after* m *steps of the Aberth method then*

$$\left\|\alpha^{(m)} - \xi\right\|_\infty \leqslant \frac{d}{2^{3m}}$$

where ξ*, is the vector of the roots and*

$$d = \frac{1}{\sqrt{n-1}} \max_{j \neq i} \left|\alpha_j - \alpha_i\right|$$

3.6 CLUSTER DETECTION AND SHIFTING TECHNIQUES

In this section will see some techniques used to detect and overcome intrinsic difficulties of roots *clusters*.

It is hard to be formal about the cluster concept. We usually call a cluster of roots a multiple root or a set of roots whose relative is distance is smaller than the machine precision.

Actually, from the point of view of the floating points computations, these concept do not differ a much.

We have already seen in Chapter 2 that most methods have only linear convergence in the case of multiple roots and, moreover, the convergence is even slower if the multiplicity of the root is high.

3.6.1 *Cluster detection*

The first problem that arises is deciding if a certain subset of roots is a cluster or not. Clearly this is even more difficult since we have not given a precise definition of cluster.

We give the following definition to make clear what we mean when speaking of clustered approximations.

DEFINITION 3.19 (Clustered approximations): Let $x_1, \ldots, x_n$ a set of approximations for the roots $\xi_1, \ldots, \xi_n$ of the polynomial $P(x)$. Let $B(x_i, r_i)$, $i = 1, \ldots, n$ be the Gerschgorin discs. The set $x_{i_1}, \ldots, x_{i_m}$, $m > 1$ is called *set of clustered approximations* if $\cup_{j=1}^{m} B(x_{i_j}, r_{i_j})$ forms a single connected component of the union $\cup_{i=1}^{n} B(x_i, r_i)$ of the Gerschgorin discs.

DEFINITION 3.20 (Isolated cluster): Under the assumptions of Definition 3.19, a set of clustered approximations $x_{i_1}, \ldots, x_{i_m}$ is called an *isolated cluster* if the set of discs $B(x_{i_j}, 3nr_{i_j})$ is disjoint from the remaining Gerschgorin discs.

We will see that the condition of isolation will be desirable to perform additional checks on the cluster structure.

In certain circumstances we use the same term *cluster* and *isolated cluster* when referring to the union of the Gerschgorin discs $\cup_{j=1}^{m} B(x_{i_j},,)r_{i_j}$.

3.6.2 *Shrinking clusters to overcome linear convergence*

As already pointed out, the worst problem that clusters introduce is a degrade in the convergence speed of the iterative methods used to approximate roots

There is little that can be done until clusters are not detected, but once we know that some approximations are likely to be m approximations of the same multiple roots, or at least approximations of close roots, we can take advantage of this knowledge to improve convergence.

Suppose that we have a set of approximations $x_{i_1}, \ldots, x_{i_m}$ that are likely to be a cluster (in the above sense). We may consider the point $\frac{1}{m}\sum_{j=1}^{m} x_{i_j}$, the mean of all these points, as the *center of the cluster*.

We have already seen in the analysis of the Durand-Kerner method that the mean of the approximations of a multiple root still has good convergence properties, and this justifies our choice of taking it as a representative of the cluster.

There is a direct way to try to identify a multiple root of order m, if it exists. Suppose that $\bar{x}$ is a root of multiplicity m of $P(x)$, i.e., there exists a polynomial $Q(x)$ such that $Q(\bar{x}) \neq 0$, and $P(x) = Q(x)(x-\bar{x})^m$. It is easy to see that $\bar{x}$ is a *simple* root of $P^{(m-1)}(x)$.

Based on this observation, we propose the following algorithm to approximate a multiple root $\bar{x}$ of a polynomial given a cluster of approximations $x_{i_1}, \ldots, x_{i_m}$.

- Consider the point $x_0 = \frac{1}{m}\sum_{j=1}^{m} x_{i_j}$ as a first approximation of $\bar{x}$;
- Apply some iterations of Newton's method to the polynomial $P^{(m-1)}(x)$ using x_0 as starting point; if the iterations converges to a point in the union of the Gerschgorin discs forming the cluster, use this value as new approximation;
- otherwise leave approximations unchanged.

In the case of a tight cluster of roots, this procedure, if working, provides an approximation of a root of $P^{(m-1)}(x)$ inside the union of discs associated with the cluster. We call this root of $P^{m)}(x)$ the *gravity center* of the cluster.

In principle, this procedure does not guarantee the convergence to a root of $P^{(m-1)}(x)$ inside the cluster. However, if the discs forming the cluster are sufficiently well separated from the remaining disc then it is possible to prove convergence. In fact the following theorem holds [MS49]

THEOREM 3.21 (Marden-Walsh): *Assume that the polynomial* $P(x)$ *has* m *zeros in* $B(c,r)$ *and* $n-m$ *zeros outside the disc* $B(c,R)$. *Then, if* $(r+R)/r > 2n/m$ *then the first derivative* $P'(x)$ *has a zero in* $B(c,r)$.

It is easy to prove that a repeated application of the Marden-Walsh theorem provides the following

COROLLARY 3.22: *Assume that the polynomial* $P(x)$ *has* m *zeros in the discs* $B(c,r)$ *and* $n-m$ *zeros outside the disc* $B(c,R)$. *If*

$$\frac{r+R}{r} > 2\frac{n!}{(n-k)!}\frac{(m-k)!}{m!}$$

for a given k, $1 \leqslant k \leqslant m-1$, *then the* k*-th derivative* $P^{(k)}(x)$ *has* $m-k$ *roots in the disc* $B(c,r)$. *In particular, for* $k = m-1$, *it follows that the* $(m-1)$st *derivative of* $P(x)$ *has one zero in* $B(c,r)$ *provided that*

$$\frac{r+R}{r} > 2\binom{n}{m-1}.$$

Indeed the above corollary provides a condition under which the derivative $P^{(m-1)}(x)$ has a simple root in the cluster formed by the Gerschgorin discs. However, the condition required by the corollary on the strong isolation of this cluster might be too restrictive in practice. In the implementation of our algorithm, that we will give next, we prefer to adopt an adaptive approach where we do not care about checking the strong isolation of clusters, and we apply a few Newton's iterations. If convergence occurs inside the cluster then our procedure is successful. If not, we skip the procedure leaving approximation unchanged. In this way we may accelerate convergence in many cases which would not be detected by the strategy based on Corollary 3.22 since the condition of strong cluster isolation is not fulfilled.

4

THE ALGORITHMS

In this chapter we present our algorithm to solve secular equations and polynomials. It is based on the ideas behind `MPSolve` by Bini and Fiorentino (see [BFoo]), the ones provided in `eigensolve` (see [Foro2] and [For]) and the progressive algorithm based on companion matrix that has been exposed in [MV95]. Below, we give an outline of the contents of this chapter.

1. A general overview of `MPSolve` is provided, with particular attention to the `MPSolve` "philosophy" that has been borrowed in `secsolve`.
2. The `secsolve` algorithm is presented in its two versions to solve secular equations.
3. It is shown that with slight modifications and with the advantage of acquiring some of the `MPSolve` infrastructure, `secsolve` can be used to solve polynomials, and that in most cases this approach can be more effective than both the original `MPSolve` and `eigensolve`.

4.1 THE MPSOLVE ALGORITHM

The `MPSolve` algorithm, developed by Bini and Fiorentino, is completely analyzed in their paper [BFoo].

Here, we review some of the main aspects of the software since it has many common elements with the implementation of `secsolve`, that we have provided.

Most of shifting and cluster detection methods have been borrowed or adapted to our implementation, and the software developed is an extensions of the original `MPSolve` implementation that can operate both in `MPSolve` mode and in `secsolve` mode.

4.1.1 *The `MPSolve` philosophy*

`MPSolve` was designed with some clear principles in mind, that helped to clarify its purpose. These principles have been followed also developing `secsolve`, so it is worth-

while to list them here. We will often refer to these ideas naming them the *MPSolve philosophy*. Here the main concepts at the base of this "philosophy" are listed.

RELATIVE ERROR ANALYSIS In MPSolve the error is always estimated by using relative error analysis, instead of absolute error. This seems to be more effective when performing floating point computations. According to this guideline our purpose will not be to find the exact roots but, given a precision of w bits, to find the roots of a polynomial whose coefficients differ from the original ones by a relative perturbation of the order of 2^{-w}.

ADAPTIVITY Instead of using the working precision needed for the worst possible input, MPSolve follows an adaptive pattern. The computation starts in standard IEEE floating point and increases the working precision *only when necessary* and *only for the roots that need it*. This allows to obtain a fine tuned algorithm that does not waste computational effort when not necessary.

IMPLICIT DEFLATION Another general rule that MPSolve follows is to use the original uncorrupted information at all the stages of the algorithm. Some algorithms may perform explicit deflation, that is when a root α is approximated to the desired precision the polynomial is deflated by dividing it by $(x - \alpha)$. The algorithm is applied recursively by using as information the coefficients of the computed quotient. Since α is not the exact root of P, an error is automatically induced on the coefficients of the quotient which propagates to the other approximations. To solve this issue MPSolve uses always the original coefficients, possibly approximated to the current precision but without losing the original information given in the input.

4.1.2 *Description of the algorithm*

The main tools used by MPSolve have already been exposed in the previous chapters, so we will now explain how is possible to assemble them to obtain the complete algorithm.

We refer to [BFoo] for a complete explanation of the procedure.

The general flow of the algorithm is composed by the following steps:

1. Compute a set of suitable starting points, either by using the Newton polygon if the coefficients are available, or by choosing them on some circles around the origin.

2. Perform Aberth iteration on the current approximations and compute Newton inclusion radii in the meantime.

3. Check if Newton radii already prove root's isolation, as seen in Section 3.5. If that's the case, jump to step 6. Otherwise, compute Gerschgorin radii as documented in Section 3.2 and update the cluster structure of the roots.

4. For each isolated cluster, compute a root $\bar{x}$ of $P^{(m-1)}$ by applying Newton's iteration starting from the arithmetic mean of the cluster approximations. Here m is the multiplicity of the cluster. If $\bar{x}$ fall outside the cluster or if Newton's iteration does not converge, leave the approximations unchanged and skip to the next cluster. Otherwise compute the first $m+1$ coefficients of $P(x-\bar{x})$ and apply the Newton polygon technique to this polynomial of degree m to place new approximations. If some approximations fall out the union of the discs forming the cluster, then leave the approximations unchanged and skip to the next cluster. If not replace the old m approximations with the new ones.

5. If the roots are not isolated or approximated to the desired precision, jump to step 2. If the roots are in the root-neighborhood for the current precision or the Newton correction is negligible with respect to the root's modulus, double the number of bits used.

6. Apply Newton iteration to the isolated roots until they are approximated with the desired number of digits.

We'll now analyze each of the steps in detail.

4.1.3 *Starting points selection*

In the original article, Aberth suggested to place the starting point on a circle of appropriate radius. In our case this method will be followed when the explicit coefficients of the polynomial are not available.

If, instead, the coefficients are known, we apply the method based on the Newton polygon exposed in Section 3.3.

Observe that both the initial approximations and the coefficients could not be representable in IEEE754 floating point. In this case we'll use another floating point type called `DPE`, i.e., Double Plus Exponent, created exactly to handle these cases. It is represented, as the name suggests, as a double number with a long integer as exponent.

4.1.4 *Aberth iterations*

Once we have the starting approximations we proceed with a packet of Aberth iterations. We apply Aberth to every approximation until one of the two following conditions is encountered:

- The approximations enters the root neighborhood of P for a suitable small ϵ. In this case every approximation is a root of a slightly perturbed polynomial, and for ϵ sufficiently small even for a polynomial with the *same floating point representation*.
- The root is isolated from all the other roots with a separation factor of $3n$. This means that the approximation is sufficiently good to make the Newton method quadratically convergent.
- The root is approximated with the required number of digits.

When an approximation satisfies one of the previous condition is marked as *approximated* and no more Aberth iterations are performed on it during this packet.

The iterations are normally performed with a Gauss-Seidel style iteration, so that when computing the Aberth correction the newer values of the approximations are always used. This often true with the exception of the cases where the algorithm is run in a parallel mode. In that case the Aberth corrections are computed in parallel, and the new approximations values are available only for the computations already terminated. In the extreme case where the number of cores is greater than the number of roots this may even degenerate in a Jacobi-like iteration, in which no new information is used.

4.1.5 *Cluster analysis*

At the end of every packet of Aberth iterations we have a set of approximations each one with an associated Newton radius. Cluster analysis, that is run at this stage, pursues two purposes:

1. Check if the Newton radii computed until now give complete isolation of the roots, in the sense that they are separated even considering the radii $3n$ times larger. As seen in Section 3.5 this proves that Newton iterations on the roots are quadratically convergent from the start.

2. If the previous condition is not verified, perform Gerschgorin cluster analysis, that allows to find the clusterization status of the roots.

If the first step is successful then no more iterations are needed. Since each approximation is guaranteed to generate a quadratic convergent Newton method to "its" root, we may proceed to the refinement step directly.

If that's not the case, we may try to see if the approximations give a separation of the root in clusters. If new clusters are found (with respect to the previous cluster analysis) and are enough separated from the others then we may perform a restart step that tries to re-dispose the approximations inside the cluster and may lead to further separation of the roots.

Once this step is completed without succeeding in Newton isolation, we have to perform other Aberth iterations.

4.1.6 *Placing refined approximations*

The analysis of the previous section has been designed to deal with multiple roots. In the case of clustered roots it may fail. Consider the polynomial with a multiple root $\bar{x}$ and a simple root $\bar{x} + \epsilon$, where ϵ is a suitably small complex number.

Intuitively, the set of $m+1$ approximations for $\bar{x}$ and $\bar{x} + \epsilon$, delivered by a numerical method, are likely to be identified as a single cluster. Moreover, the process outlined in the previous section may not work since the Newton iteration performed on $P^{(m-1)}(x)$ may converge to something which is closer to $\bar{x} + \epsilon$ than to $\bar{x}$.

Our proposed workaround to this case is to apply the starting criterion already presented in section 3.3 to place starting points along circles centered at the root of $P^{(m-1)}(x)$ inside the cluster. We have already discussed that this method is particularly effective in recognizing small and big roots. Since the mean of the approximations will be near to $\bar{x}$ the roots $\bar{x}$ and $\bar{x} + \epsilon$ have quite different moduli if shifted in the center of the cluster ($\bar{x}$ will be small with respect to $\bar{x} + \epsilon$).

Moreover it is possible to re-apply cluster analysis on the repositioned roots and actually detect the inner cluster. Consider the following algorithm:

- Compute the root $\bar{x}$ of $P^{(m-1)}(x)$ as shown in Section 3.6.2;
- if $\bar{x}$ is in the cluster compute the first $m+1$ coefficients of the polynomial $P(x-\bar{x})$, otherwise leave the approximation unchanged and exit;

- Apply the Newton polygon strategy to this polynomial of degree m, and displace new approximations in the related circles.
- If the new approximations are inside the union of the Gerschgorin discs forming the cluster then output them; otherwise leave the approximations unchanged and exit.

REMARK 4.1: A possible strategy to continue the approximation procedure is to reposition the roots on circles with the radii obtained by the application of the criterion seen in section 3.3.

4.1.7 *Identifying multiple roots*

Consider the case where we have a cluster with a multiple root and a simple root near to it. It would be nice to have some method to detect when a cluster contains additional roots that are simple and relatively far from the real problematic situation but the Gerschgorin discs are not small enough to separate them.

Observe that, if one suspect that a root is of this kind, a possible strategy to detect this situation is *temporary removing* it from the cluster, and then performing cluster analysis as seen before. This procedure may give some radii inclusions obtained through the Newton polygon that confirm our hypothesis that the separated root is not really part of that cluster.

We will see in the numerical experiments that this strategy is quite effective in practice, but the immediate problem that arises is to decide which criterion shall be used to detect these roots.

A possible criterion is based on the observation that the real purpose of cluster analysis is overcoming the convergence slowdown in approximating multiple roots or numerically multiple roots.

Consider, as an example, the Kirinnis polynomial of degree 44. This is the polynomial with integer coefficients that has 1, -1, i and $-i$ as multiple roots of order 10 and then 4 simple roots that are near these ones, precisely $1+\epsilon$, $-1-\epsilon$, $(1+\epsilon)i$ and $(-1-\epsilon)i$ where $\epsilon \approx \frac{1}{1024}$.

This is a difficult polynomial to solve since the simple roots near the multiple ones degrade the effectiveness of our cluster analysis techniques. We expect a multiple root's approximation to have approximately $\frac{u}{m}$ correct digits where u are the digits available in the current precision and m is the order of the multiple roots. If we find a root in a cluster that has at least $\frac{u}{2}$ correct digits (according to the inclusion radii seen

in Corollary 3.3) then we flag it as quasi-approximated, and we try to detach it from the cluster as seen above.

If the detachment process does not yield separation of the root from the rest of the cluster we choose to add it back.

4.1.8 *Refinement step*

Once the roots are Newton isolated, we have two choices. If the user has asked only for isolated approximations (that is the default goal of `MPSolve`) we terminate the algorithm. Otherwise we have to apply the Newton method to every approximation, until the required number of correct digits are found. This is quite easy to control since we know by the results of Section 3.5 that the convergence is quadratic. It may be paid some attention, though, to the working precision of the roots since we have to make sure that floating point errors are negligible with respect to the current error on the approximations.

4.2 OUTLINE OF SECSOLVE

This section will cover the description and implementation of the `secsolve` algorithm. This is the algorithm that is used to approximate the roots of secular equation, and that can be generalized to approximate polynomial roots.

All the tools needed to describe and understand this algorithm have been developed and studied in the previous chapters.

We start by exposing the idea behind `secsolve`. Suppose that we want to approximate the roots of the secular equation

$$S(x) = \sum_{i=1}^{n} \frac{a_i}{x - b_i} - 1 = 0.$$

We've already seen in Section 1.2 that these roots coincide with the ones of the monic polynomial $P(x) = -\prod_{i=1}^{n}(x - b_i)S(x)$ of degree n. So our idea for achieving the approximation of the roots is applying the Aberth method to $P(x)$ given a_i, b_i, $i = 1, \ldots, n$.

4.2.1 *The first implementation*

We have studied two ways of pursuing this approach, and the first one is quite straightforward. It consists of computing the Newton correction $N(x) = \frac{P(x)}{P'(x)}$ associated with P without computing the coefficients of P explicitly, but using the procedure already discussed in Section 1.6.

Once that we have the procedure to compute $N(x)$, applying Aberth method is quite easy and can be achieved following what has been exposed in Section 2.4, and in particular in Remark 2.1.

So one could build an algorithm following substantially the `MPSolve` algorithm, but using the implicit Aberth computation.

The only open problem, at this point, is finding a good criterion to choose the initial starting points.

We know that the secular roots are contained in Gerschgorin discs that have b_i as centers as seen in Section 1.7. Based on this observation, one may wonder if the b_i are a good choice for starting approximations.

It turns out they are, and that this choice works well. It does even more in the second version of the `secsolve` algorithm, where this is a really good reason to choose the b_i as starting points.

To be more precise, we may note that choosing *exactly* b_i as starting points may seem little unfortunate since they correspond to poles of the secular function $S(x)$. Actually that's not a problem since we have developed an alternative algorithm for the computation of the Newton correction at b_i (see Section 1.6.3 for the details). In fact, even if $S(x)$ is singular at b_i there's no reason why this should be true even for the polynomial associated with it.

4.2.2 *Modified algorithm with regeneration*

We explain now the second version of `secsolve` that can be seen as an extension of the first one.

The idea on which this algorithm is built is that, as we have previously seen in section 1.5.2, a secular equation is better conditioned if the nodes b_i are better approximations of the roots.

Clearly it is not easy to have good approximations from the start, but we may obtain them during the algorithm execution. So these can be used to compute *another* secular equation with the same roots as seen in Section 1.4.

So the basic idea is to follow this procedure:

1. Compute a set of initial approximations.

2. Start Aberth iterations applied to $P(x) = -\prod_{i=1}^{n}(x - b_i)S(x)$ until we find that the approximations are in the root neighborhood, or the Newton correction become sufficiently small compared to the current precision. The necessary check used to determine if an approximation belongs to the root neighborhood of $S(x)$ has been exposed in Section 1.3. The iterations are performed only on the components that are not yet approximated or Newton isolated.

3. Use the computed approximations as new nodes for another secular equation, and restart from point 2 until convergence, or isolation. The algorithms for the regeneration of an equivalent secular equation starting from the polynomial or another secular equation have been documented in Section 1.4.

4. Perform cluster analysis. We use the same cluster analysis routines already used for `MPSolve`, with some small changes to reflect the new structure of the problem. If some clusters are well isolated by the others (see Section 3.6.1) we perform the shift of the polynomial in the center if the coefficients of the polynomial are available.

REMARK 4.2: Recall that, in the classic `MPSolve` algorithm we have to perform Gerschgorin cluster analysis at the end of each iteration packet, and so we must compute the Gerschgorin inclusions radius as seen in Section 3.2. Note that if we compute the $\hat{a}_i$ for the new representation on the nodes x_i, where x_i are the approximations at the end of the iteration packet, then the Gerschgorin inclusion radii are $n|\hat{a}_i|$. There seems to be no overhead in computing the new representation. That is not completely true because in this case the a_i must be computed in high precision while the evaluation in `MPSolve` can be carried out in standard floating point (with a proper error analysis).

There are some issues that should be discussed before going on and implementing this algorithm. The first is that we need some kind of precise analysis of the error on the polynomial evaluation in order to be able to guarantee that the coefficients of the new secular equation are correct to the current precision.

Actually, managing precision is one of the more subtle things of this algorithm and must be kept under control. These types of issues are the topic of the next section.

4.3 MANAGING PRECISION

All the implementation and the design of `MPSolve` and `secsolve` is done in a multi-precision framework.

We have already seen rigorous error bounds on the inclusion theorem stated in the previous chapters, that will help to analyze this problem.

The approaches followed by `MPSolve` and `secsolve`, regarding precision management, are quite different.

The first works at increasing levels of precisions, paying attention to increase the precision only on the roots that need it. This means that clustered or bad conditioned roots will use a higher precision while the good conditioned ones will not.

More precisely, Aberth iterations are applied until roots have converged, fall in the root neighborhood or lead to a Newton correction whose modulus is relatively smaller than the machine precision. Once this goal is reached and all other steps listed in the previous section have been applied the bits of precision used to represent the root are doubled. Aberth iterations are then applied to the roots that haven't still reached the convergent state.

The `secsolve` algorithm, instead, follows a different approach. It uses two different states that are continuously alternated:

REGENERATION STAGE High precision is used to generate a secular equation on some given nodes b_i with the same roots of the original polynomial or secular equation. The precision in this step is dynamically determined on every root to make sure that the values of a_i are computed with a relative error smaller than the floating point precision.

The error on the computation of the a_i is estimated with the error bounds presented in Section 1.4.

ABERTH STAGE After the regeneration step Aberth iterations are applied implicitly until the approximations fall in the root neighborhood. Once this happen a new representation is computed with the previous approach. If in the Aberth stage the stop condition guarantees a good approximation of the roots in the current precision the number of bits used in the representation are doubled. This happens if one of the following two criterion is verified:

- The approximation falls in the root neighborhood and the conditioning is smaller than 1. This implies that the small error on the coefficients is greater than the actual error on the approximation.

- The modulus of the Newton correction is smaller than the product of the machine precision and the modulus of the approximation and computed accurately. As seen in Section 3.1 this implies that the approximations are accurate.

The approach of `secsolve` has the advantage that even in case of bad conditioned or multiple roots it may be possible to perform the Aberth iteration in floating point, "wasting" computational effort only in a single step at the start of the iteration.

Estimating the necessary precision for regeneration

When regenerating the secular equation, i.e., computing the new values of a_i, we would like to estimate the necessary precision that must be used to perform the computation.

Recall that, as seen in Section 1.4, we have a total error on the regenerated coefficients given by

$$\mathrm{fl}(a_i) = a_i(1 + \delta_i + \sum_{\substack{k=1 \\ k\neq i}}^{n} (\epsilon_i + \gamma_i))$$

where δ_i is the error on the evaluation of $P(b_i)$ (that may be obtained by Horner or by an implicit evaluation of P), $|\epsilon_i| \leqslant \epsilon_\pm$ and $|\gamma_i| \leqslant \epsilon_*$.

Let u be the relative machine precision in floating point with the number of bits used for then Aberth iteration.

We would like to have $|\delta_i| \leqslant u$ and $\sum_{\substack{k=1 \\ k\neq i}}^{n} |\epsilon_i + \gamma_i| \leqslant u$. For the first term we rely on the algorithm used to evaluate the polynomial and determine the minimum number of bits necessary for the computation. For the second term, since $|\epsilon_i| \dot{\leqslant} u'$ and $|\gamma_i| \dot{\leqslant} 2\sqrt{2}u'$ where u' is the machine precision used in the regeneration step, we may simply choose u' so that the relation

$$\sum_{\substack{k=1 \\ k\neq i}}^{n} |\epsilon_i + \gamma_i| \dot{\leqslant} n(1 + 2\sqrt{2})u' \leqslant u$$

holds. The number of bits is chosen then as the maximum of the two number of bits obtained by the two estimates.

5

COMPUTATIONAL ISSUES

5.1 IMPLEMENTATION

The algorithm described in this thesis has been implemented by extending the capabilities of the pre-existing software `MPSolve`, written by Bini and Fiorentino and documented in [BF].

`MPSolve` has been modified in several ways through the process

- Some of the ideas exposed here that were not present in the original version have been added to the algorithm. They are documented in Section 5.1.1.
- The old algorithm has been adapted to solve secular equations, creating a new mode in which `MPSolve` can be run.
- A new algorithm has been introduced that exploits regeneration techniques exposed in Chapter 4. This can be used to solve both polynomials and secular equations.

The final version of the `MPSolve` package will appear soon at `http://www.dm.unipi.it/cluster-pages/mpsolve/`.

5.1.1 *Extensions to the original algorithm*

Some of the new ideas presented in this thesis have been introduced to complete the existing algorithm.

STARTING POINT PLACEMENT The routine for the placement of the starting points has been modified to compute the right shifts to the circles that maximizes the distance between the starting points, and to collapse the circles that are too near, as documented in Section 3.3.3.

CLUSTER ANALYSIS The cluster analysis has been modified to use Gerschgorin circles instead of Newton ones, and the new strategy for the empirical removal of quasi-

approximated roots in the cluster has been added. It has seen to be effective on polynomials such as the one of Kirinnins.

5.1.2 *Implementation of the new algorithm*

As a final step the new algorithm with the regeneration step described in Chapter 4 has been implemented and is able to solve both secular equations and polynomials.

It is possible to select the algorithm to use for the resolution with the `-a` switch at command line.

5.1.3 *Parallelization*

An additional modification that has been done to the original code is the parallelization of some parts of the algorithm.

Precisely, both the Aberth iterations and the regeneration step have been parallelized. Parallelizing the regeneration is not too difficult since all the computations to obtain the new values of a_i are independent. They can be carried out in parallel with the only knowledge of the initial data, i.e., the values of the new b_i and the old a_i and b_i.

Parallelizing the Aberth iterations is a slightly more delicate task. We have already discussed that we would like to use a Gauss-Seidel style iterations, i.e., we would like to use always the more updated values of the approximations x_i to compute the Aberth correction.

This is not possible, in general, if the computation for the Aberth corrections are carried out in parallel. We have made the choice of computing these values in a semi Gauss-Seidel style. Precisely, we use the more updated values that are available, i.e., all the ones that have already been computed.

In a general i-th step the knowledge of the previous updated approximations is not guaranteed, but if k processes are used it is likely to know the value of $i - k$ approximations (if $i > k$). If the degree of the polynomial is much bigger than the number of processor available this is similar to a real Gauss-Seidel iteration. In the extreme case where the number of processors is greater than the degree, instead, this is exactly a Jacobi iteration.

This change of strategy does not seem to impact much the performance, and is quite convenient given the acceleration offered by the parallel environment.

5.1.4 *Input format*

The compatibility with the old format used by `MPSolve` has been maintained, but since an extension was needed to cope with secular equations a new format has been introduced.

An input file for `MPSolve` has the following structure

```
! Comments are prefixed with the symbol !
[ Preamble ]

[ Coefficients ]
```

Where [Preamble] is a list of commands or assignment that specify the characteristic of the polynomials, and [Coefficients] is the list of the coefficients according to the format specified above. All the commands must be terminated by a semicolon.

The available commands are listed here

COMPLEX The polynomial coefficients are complex. This is the default and implies that the coefficients will be given as a pair of number representing the real and imaginary part.

REAL The polynomial coefficients are real. This implies that only the real part is needed in the coefficients list.

INTEGER The coefficients are integer.

RATIONAL The coefficients are rational and will be specified in the form a/b or a.

FLOATINGPOINT The polynomials are floating point numbers, and so will be listed in the standard notation 1.234e5. Integer input is also accepted.

DEGREE This is a keyword that needs a value and sets the degree of the polynomial. For example, **Degree**=5;

MONOMIAL The input is given in the monomial basis, and so $n+1$ coefficients will be inserted ($2n+2$ if the polynomial is complex) where n is the degree of the polynomial. The terms must be listed starting in increasing degree order.

SECULAR The input is given as a secular equation. The coefficients a_i and b_i need to be specified and need to be listed in pairs a(i) b(i). See the following for a simple example.

PRECISION Set the bits of precision of the input coefficients.

DENSE The input polynomial has dense coefficients.

SPARSE The input polynomial has sparse coefficients and so the coefficients must be inserted with their degree.

5.1.5 *Input examples*

We report here some examples of input files in order to clarify the explanation of the previous section.

Consider the polynomial $x^3 - 1$. We have the following input file

```
! The polynomial x^3 - 1
Monomial;
Integer;
Real;
Degree=3;

1 0 0 -1
```

The same polynomial may be written in its sparse representation where only the non zero coefficients are reported, together with their degree. This has advantages both in compactness of the code and in performance, since the sparse evaluation algorithms are used.

```
! x^3 - 1 in its sparse representation
Monomial;
Integer;
Real;
Sparse;
Degree=3;

3 1
0 -1
```

Every row is in the form `Degree Value`. The representation for secular equation is exactly the same with the exception that we have the double number of coefficients for the same degree. These are expressed in the form `a(i) b(i)`. So the secular equation

$$\frac{1}{x-2} - \frac{4}{x-5} - 1 = 0$$

is described with the following input file:

```
Secular;
Integer;
Real;
Degree=2;

1 -2
-4 -5
```

Consider now an example of polynomial with complex coefficients: $x^3 - 2ix + 1$. The input file for this polynomial is

```
Monomial;
Integer;
Complex;
Degree=2;

1 0
0 -2
1 0
```

5.2 NUMERICAL EXPERIMENTS

In this section some numerical experiments with our algorithm are reported.

To better understand where `secsolve` excels or is not so good we have performed a side to side comparison with the old `MPSolve` version and with `eigensolve`, a polynomial rootfinder developed by Fortune and presented in [Foro2].

We analyze the following class of polynomials:

ROOTS OF UNITY These are the polynomials $x^n - 1$ for various values of n. Since these polynomials are sparse we expect that methods that can exploit this structure will be faster.

MANDELBROT These polynomials have integer coefficients. They can be obtained by by a recurrent relation on the degree. Let P_d the polynomial of degree d. We have

$$\begin{cases} P_0(z) = 1 \\ P_{d+1}(z) = zP_d^2 + 1 \end{cases}.$$

The roots of the polynomial of degree 1023 are shown in Figure 5.1.

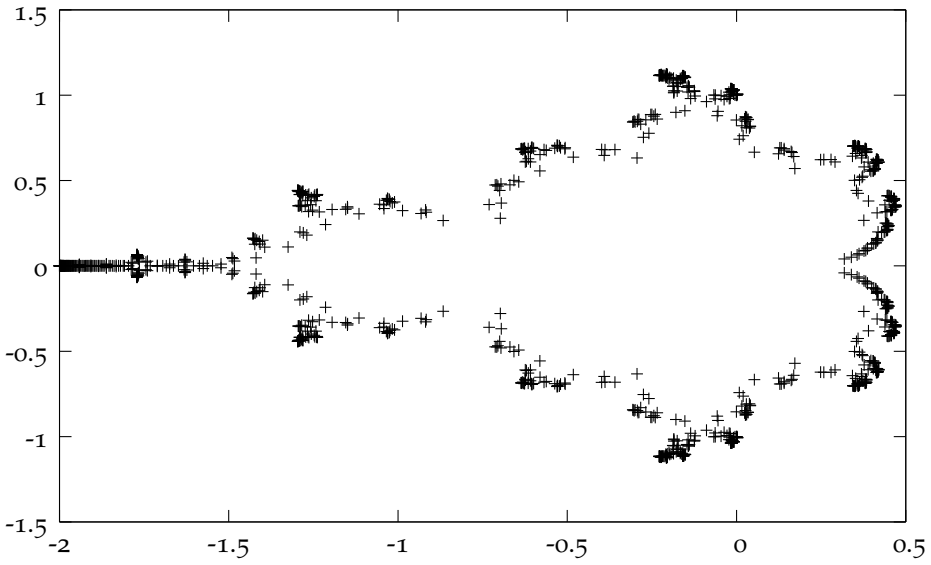


Figure 5.1.: Roots of the Mandelbrot polynomial of degree 1023

ORTHOGONAL POLYNOMIALS Some families of orthogonal polynomials are analyzed. Precisely, Chebyshev, Legendre, Hermite and Laguerre's polynomials will be tested.

CHROMATIC Some kinds of chromatic polynomials are tested. Their input files start with `chrmc`. The roots of one of these polynomials can be seen in Figure 5.2.

LARGE AND SMALL ROOTS This class of polynomial have both small and big (in modulus) roots, and have been designed precisely for testing how rootfinding algo-

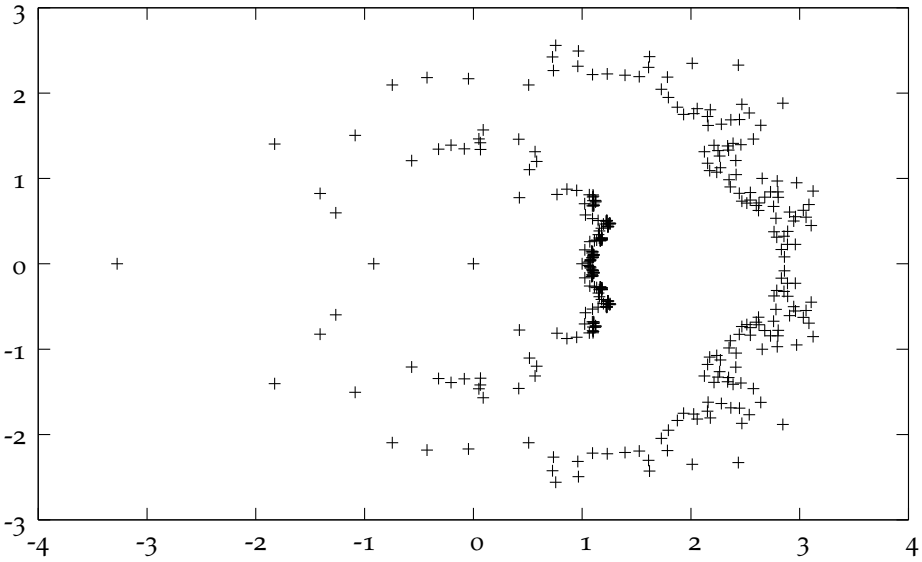


Figure 5.2.: The roots of a chromatic polynomial of degree 342

rithms deal with these extreme cases. Their input files are named `lar_n.pol` in the case of large roots and `lsr_.pol` in the case of both large and small roots.

PARTITION These polynomials have as k-th coefficient the number of different ways in which k can be obtained as a sum of positive integers. The roots of these polynomials converge to a path when $n \to \infty$ as is recognizable in Figure 5.3.

WILKINSON This family of polynomials have been traditionally considered a difficult one for their bad conditioning properties. They were introduced by Wilkinson in [Wil59] and have the integers from 1 to n as roots, where n is the degree of the polynomial.

KIRINNIS These polynomials have 1, -1, i and $-i$ as multiple roots and 4 simple roots near to these. This family of polynomials is very useful to benchmark cluster analysis capabilities of an algorithm, since the case of a multiple root with a simple root near to it is usually quite hard to detect.

MIGNOTTE Mignotte polynomials have integer coefficients and the characteristic of having two roots whose distance is close to Mahler's bound.

SPIRAL Spiral polynomials are a class of polynomials defined by the formula

$$P(x) = (x+1)(x+1+\epsilon)(x+1+\epsilon+\epsilon^2)\dots(x+1+\epsilon+\dots+\epsilon^n)$$

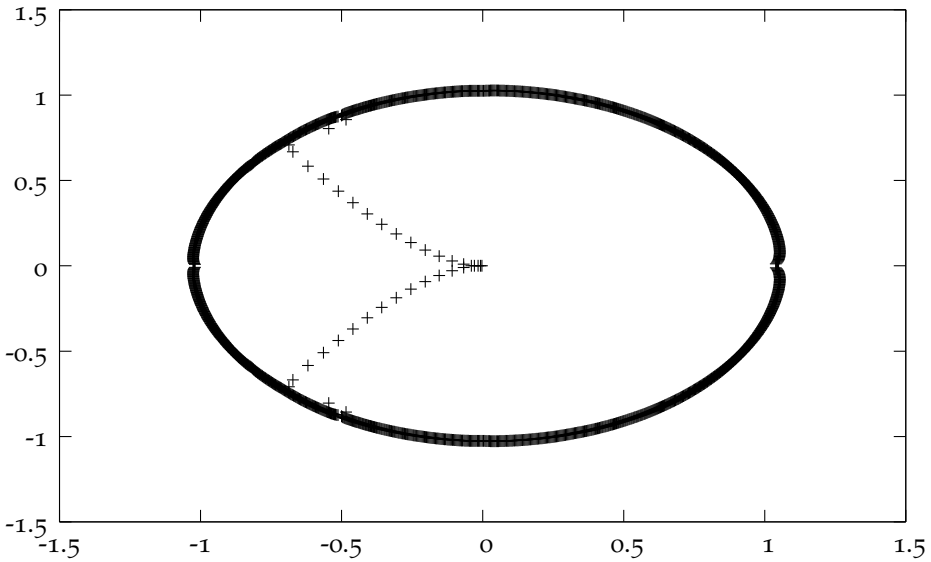


Figure 5.3.: The roots of the partition polynomial of degree 1600

for every n. They are useful to benchmark the capabilities of an algorithm to extract approximations of the roots at various precisions.

TRUNCATED EXPONENTIAL The series of the exponential truncated to the n-th element, with various values of n, are considered as test polynomials.

To correctly evaluate the performance of the various algorithms we chose the same goal for the computation. The default goal for `MPSolve` is to give Newton isolated approximations or, if not possible withing the maximum specified floating point digits, to give approximated roots. The default for `eigensolve`, instead, is to give approximated roots so we chose this last target also for `MPSolve`.

The results of the multithread runs are also reported but are not directly comparable to the other ones because of some automatic overclocking that happens when only a processor is used. The tests have been run on a system with two 6-core processors and Hyperthreading ™.

The result of the tests have been reported in Table 5.1. The fastest single threaded solver in every row is marked with bold font.

We can observe that `secsolve` is faster in a good amount of test polynomials. It is notable slower in the case of the roots of unity with respect to `MPSolve` 2.2, since the latter can take fully advantage of the sparse representation of $x^n - 1$, while `secsolve`

	MPSolve 2.2	eigensolve	secsolve	MPSolve 3.0 (mt)	secsolve (mt)
nroots800	**0.11**	8.37	1.18	0.11	0.38
nroots1600	**0.34**	57.44	4.41	0.27	1.27
chebyshev80	0.09	**0.08**	0.12	0.11	0.03
chebyshev160	0.97	0.69	**0.37**	0.69	0.08
chebyshev320	9.36	7.00	**2.32**	5.66	0.30
hermite80	**0.05**	0.06	0.11	0.07	0.03
hermite160	0.55	0.54	**0.35**	0.50	0.07
hermite320	5.33	5.51	**1.70**	3.57	0.21
chrma342	19.56	4.43	**3.89**	9.79	0.47
chrmad340	29.71	**4.46**	4.82	13.00	0.63
exp100	0.29	0.14	**0.12**	0.14	0.04
exp200	1.05	1.19	**0.59**	0.93	0.08
exp400	10.28	10.67	**8.82**	1.45	0.95
mand127	0.30	0.21	**0.18**	0.34	0.05
mand255	2.82	2.24	**1.08**	2.34	0.17
mand511	31.18	20.08	**7.04**	14.68	0.83
mand1023	456.91	229.60	**52.11**	93.60	6.17
mand2047	11158.72	3860.10	**517.32**	1054.67	46.21
spiral20 (50 digits)	0.70	**0.07**	0.34	0.54	0.21
spiral20 (1000 digits)	0.76	**0.13**	0.60	0.55	0.25
mig1_200 (50 digits)	**0.04**	0.85	0.32	0.11	0.19
mig1_200 (1000 digits)	**0.15**	3.18	2.18	0.18	0.19
kir1_10 (100 digits)	0.23	**0.06**	0.37	0.56	0.20
kir1_10 (1000 digits)	7.23	**1.50**	5.33	9.35	2.24
kir1_10 (4000 digits)	55.30	77.38	**43.23**	70.89	17.01
partition400	**0.67**	4.81	1.02	0.74	0.17
partition800	5.49	36.90	**4.92**	4.33	0.63
partition1600	38.45	390.62	**25.38**	21.86	3.41
partition3200	213.34	7038.98	**111.04**	109.12	13.26
partition6400	1303.83	45953.33	**408.78**	470.37	52.02
partition12800	7845.62	-	**2786.65**	2369.62	291.60

Table 5.1.: Timings of the test runs of MPSolve 2.2, eigensolve and secsolve. The columns marked with (mt) refer to the tests with multithreading enabled.

uses a dense secular representation. The same holds for the Mignotte-like polynomials that have a sparse representation.

A

ERROR ANALYSIS IN FLOATING POINT

This appendix has the purpose of making light in some error analysis performed in the thesis. Giving the exact error bounds in floating computations might be tricky; this is particularly true when complex arithmetic is involved since (most) computers don't have a logic unit that can handle complex numbers directly.

We start by fixing some notation and conventions:

- u denotes the machine precision. In the case of double precision arithmetic this constant takes the value $u = 2^{-53} = 10^{-16}$. In general, working with d binary digits, it holds that $u = 2^{1-d}$.

- In real arithmetic for every operation op in $\{+, /, -, \cdot\}$ the following relation holds:

$$\mathrm{fl}\,(x \text{ op } y) = (x \text{ op } y)(1+\delta) \qquad |\delta| \leqslant u$$

A detailed analysis of this argument can be found on [Hig96].

A.1 BASIC OPERATIONS ON THE COMPLEX FIELD

Arithmetic on the complex field will be based on real floating point arithmetic using the natural representation $z = a + ib$, for $a, b \in \mathbb{R}$, where i denotes the imaginary unit such that $i^2 = -1$.

The sum $z_3 = z_1 + z_2$ of two complex numbers $z_1 = a_1 + ib_1$ and $z_2 = a_2 + ib_2$ is trivially computed by means of

$$z_3 = (a_1 + a_2) + i(b_1 + b_2).$$

The product $z_3 = z_1 * z_2$ is computed with the algorithm based on 4 real multiplications and 2 real additions

$$z_3 = (a_1 * a_2 - b_1 * b_2) + i(a_1 * b_2 + a_2 * b_1).$$

The algorithm for computing the quotient $z_3 = z_1/z_2$ relies on computing the reciprocal of z_2 and on multiplying it by z_1. We refer the reader to the book [Hig96].

Performing complex floating point arithmetic with the algorithm we have the following properties.

THEOREM A.1: *Given the model above for complex arithmetic, the following bounds are valid for the standard operations.*

$$\mathrm{fl}(x \pm y) = (x \pm y)(1+\delta) \qquad |\delta| \leqslant u \tag{A.1}$$

$$\mathrm{fl}(xy) = (xy)(1+\delta) \qquad |\delta| \leqslant \sqrt{2}\frac{2u}{1-2u} \tag{A.2}$$

$$\mathrm{fl}\left(\frac{x}{y}\right) = \left(\frac{x}{y}\right)(1+\delta) \qquad |\delta| \leqslant \sqrt{2}\frac{7u}{1-7u} \tag{A.3}$$

These bounds will be called $\epsilon_{\pm}$, ϵ_{*} and $\epsilon_{\div}$, respectively.

We report the error bound in performing the summation $\sigma = \sum_{i=1}^{n} x_i$ of n numbers. we consider two different algorithms. The first one computes σ sequentially, the second one recursively

Algorithm 6 Algorithm for the sequential evaluation of σ

```
1: procedure SequentialSum(x)
2:     σ ← 0
3:     for i = 1 : n do
4:         σ ← σ + x_i
5:     end for
6:     return σ
7: end procedure
```

Both the algorithms perform the same number of arithmetic operations, moreover they are backward stable as stated by the following

THEOREM A.2: *Let* $\mathrm{fl}(\sigma_{\mathrm{sec}})$ *and* $\mathrm{fl}(\sigma_{\mathrm{rec}})$ *the values delivered by the sequential and the recursive summation algorithms performed in floating point arithmetic with complex numbers. Then*

$$\begin{aligned} \mathrm{fl}(\sigma_{\mathrm{sec}}) &\doteq \textstyle\sum_{i=1}^{n} x_i(1+\epsilon_i) \quad |\epsilon_i| \leqslant (n-i+1)\epsilon_{\pm} \\ \mathrm{fl}(\sigma_{\mathrm{sec}}) &\doteq \textstyle\sum_{i=1}^{n} x_i(1+\delta_i) \quad |\delta_i| \leqslant \lceil \log_2 n \rceil \epsilon_{\pm} \end{aligned} \tag{A.4}$$

Algorithm 7 Algorithm for the sequential evaluation of σ

```
1: procedure RecursiveSum(x)
2:     if n = 1 then
3:         σ ← x_1
4:     else
5:         if n=2 then
6:             σ ← x_1 + x_2
7:         else
8:             σ_1 ← RecursiveSum(x_1, x_3, ...)
9:             σ_2 ← RecursiveSum(x_2, x_4, ...)
10:            σ ← σ_1 + σ_2
11:        end if
12:    end if
13:    return σ
14: end procedure
```

BIBLIOGRAPHY

[Abe73] O. Aberth. Iteration methods for finding all zeros of a polynomial simultaneously. *Mathematics of Computation*, 27(122):339–344, 1973.

[Bai20] L. Bairstow. *Applied aerodynamics*. Longmans, Green and co., 1920.

[BF] DA Bini and G. Fiorentino. Numerical computation of polynomial roots using mpsolve version 2.2 (january 2000). *Software package and documentation available for download at ftp://ftp. dm. unipi. it/pub/mpsolve*.

[BF00] D.A. Bini and G. Fiorentino. Design, analysis, and implementation of a multiprecision polynomial rootfinder. *Numerical Algorithms*, 23(2):127–173, 2000.

[BG07] R.P. Boyer and W.M.Y. Goh. Partition polynomials: asymptotics and zeros. *Arxiv preprint arXiv:0711.1373*, 2007.

[BGP04] DA Bini, L. Gemignani, and VY Pan. Inverse power and durand-kerner iterations for univariate polynomial root-finding. *Computers & Mathematics with Applications*, 47(2-3):447–459, 2004.

[Bin96] D.A. Bini. Numerical computation of polynomial zeros by means of aberth's method. *Numerical Algorithms*, 13(2):179–200, 1996.

[BNS78] J.R. Bunch, C.P. Nielsen, and D.C. Sorensen. Rank-one modification of the symmetric eigenproblem. *Numerische Mathematik*, 31(1):31–48, 1978.

[BS63] W. Börsch-Supan. A posteriori error bounds for the zeros of polynomials. *Numer. Math.*, 5:380–398, 1963.

[Car91] C. Carstensen. Linear construction of companion matrices. *Linear Algebra and Its Applications*, 149:191–214, 1991.

[CGM80] R A Cuninghame-Green and P F J Meijer. An algebra for piecewise-linear minimax problems. In *Discrete Applied Mathematics,2:267–294*, 1980.

[Dem] J.W. Demmel. Applied numerical linear algebra. 1997. *SIAM, Philadelphia, PA*.

[EH12] Yuli Eidelman and Iulian Haimovici. Divide and conquer method for matrices with quasiseparable representations. *Presented at Structured Numerical Linear and Multilinear Algebra Problems Analysis, Algorithms, and Applications, KU Leuven*, September 2012.

[Ehr67] L. W. Ehrlich. A modified newton method for polynomials. *Commun. ACM*, 10(2):107–108, February 1967.

[FGHO97] R.D. Fierro, G.H. Golub, P.C. Hansen, and D.P. O'Leary. Regularization by truncated total least squares. *SIAM Journal on Scientific Computing*, 18(4):1223, 1997.

[For] S. Fortune. Convergence analysis of an iterated-eigenvalue polynomial root-finding algorithm. In *submitted to workshop proceedings*.

[For02] S. Fortune. An iterated eigenvalue algorithm for approximating roots of univariate polynomials. *Journal of Symbolic Computation*, 33(5):627–646, 2002.

[Fra91] P. Fraigniaud. The durand-kerner polynomials roots-finding method in case of multiple roots. *BIT Numerical Mathematics*, 31(1):112–123, 1991.

[Gan80] W. Gander. Least squares with a quadratic constraint. *Numerische Mathematik*, 36(3):291–307, 1980.

[GGvM89] W. Gander, G.H. Golub, and U. von Matt. A constrained eigenvalue problem. *Linear Algebra and its applications*, 114:815–839, 1989.

[Gol73] G.H. Golub. Some modified matrix eigenvalue problems. *Siam Review*, pages 318–334, 1973.

[Gra72] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information processing letters*, 1(4):132–133, 1972.

[GS09] S. Gaubert and M. Sharify. Tropical scaling of polynomial matrices. *Positive systems*, pages 291–303, 2009.

[GVL80] G.H. Golub and C.F. Van Loan. An analysis of the total least squares problem. *SIAM Journal on Numerical Analysis*, pages 883–893, 1980.

[Han62] DC Handscomb. Computation of the latent roots of a hessenberg matrix by bairstow's method. *The Computer Journal*, 5(2):139–141, 1962.

[Hen88] P. Henrici. *Applied and Computational Complex Analysis, Power Series Integration Conformal Mapping Location of Zero*. Wiley Classics Library. Wiley, 1988.

[Hig96] N.J. Higham. *Accuracy and stability of numerical algorithms*. Number 48. Siam, 1996.

[Luk96] WS Luk. Finding roots of a real polynomial simultaneously by means of Bairstow's method. *BIT Numerical Mathematics*, 36(2):302–308, 1996.

[McN07] J.M. McNamee. *Numerical methods for roots of polynomials*, volume 14. Elsevier Science, 2007.

[MMMM06] D. Steven Mackey, Niloufer Mackey, Christian Mehl, and Volker Mehrmann. Structured polynomial eigenvalue problems: good vibrations from good linearizations. *SIAM J. Matrix Anal. Appl.*, 28(4):1029–1051 (electronic), 2006.

[MS49] M. Marden and American Mathematical Society. *The Geometry of the Zeros of a Polynomial in a Complex Variable*, volume 3. American mathematical society New York, 1949.

[MV95] F. Malek and R. Vaillancourt. A composite polynomial zerofinding matrix algorithm. *Computers & Mathematics with Applications*, 30(2):37–47, 1995.

[Sha11] M. Sharify. *Scaling Algorithms and Tropical Methods in Numerical Matrix Analysis*. PhD thesis, ÉCOLE POLYTECHNIQUE, 2011.

[Til98] P. Tilli. Convergence conditions of some methods for the simultaneous computation of polynomial zeros. *Calcolo*, 35(1):3–15, 1998.

[Wil59] JH Wilkinson. The evaluation of the zeros of ill-conditioned polynomials. part i. *Numerische Mathematik*, 1(1):150–166, 1959.