Noname manuscript No.
(will be inserted by the editor)

An Effective Implementation of a Modified Laguerre Method for the Roots of a Polynomial

Thomas R. Cameron

Received: date / Accepted: date

Abstract Two common strategies for computing all roots of a polynomial with Laguerre's method are explicit deflation and Maehly's procedure. The former is only a semi-stable process and is not suitable for solving large degree polynomial equations. In contrast, the latter implicitly deflates the polynomial using previously accepted roots and is, therefore, a more practical strategy for solving large degree polynomial equations. However, since the roots of a polynomial are computed sequentially, this method cannot take advantage of parallel systems. In this article, we present an implementation of a modified Laguerre method for the simultaneous approximation of all roots of a polynomial. We provide a derivation of this method along with a detailed analysis of our algorithm's initial estimates, stopping criterion, and stability. Finally, the results of several numerical experiments are provided to verify our analysis and the effectiveness of our algorithm.

Keywords Laguerre's method · polynomial roots · mathematical software

Mathematics Subject Classification (2010) 26C10 · 65H04 · 65Y20

1 Introduction

In the mid-1800's, Laguerre first published his root-finding method in *Racines d'une Équation Algébrique* and it was later republished in *Oeuvres de Laguerre* in 1898 [21]. Since then, research on modifications, generalizations, and analysis of Laguerre's method has permeated the mathematical literature [10,12, 17,20,24,34,36,38]. In addition, Laguerre's method has been applied to the matrix and polynomial eigenvalue problem [15,22,23,30].

Thomas R. Cameron
Davidson College
Mathematics and Computer Science Department
Tel.: +1 (704)-894-2241
E-mail: thcameron@davidson.edu

Despite a significant amount of attention in the literature, we are only aware of two popular software packages that employ Laguerre's method to solve for the roots of a polynomial: *zroots* from Numerical Recipes and *C02AFF* from the NAG library [13,26,28,35]. Both *zroots* and *C02AFF* use forward deflation to ensure that subsequent iterations tend to a different root. However, *zroots* does nothing to ensure that the roots are computed in ascending order of absolute value, which is vital for the backward stability of forward deflation [40]. In contrast, the NAG library routine *C02AFF* uses a modified Laguerre method that was first proposed by Smith and guarantees the roots are computed in ascending order of absolute value [38]. For this reason, the routine *C02AFF* is more reliable than *zroots*.

Modern-day polynomial root problems, such as those that stem from computer algebra applications or digital signal processing, require software that can solve for the roots of a polynomial of degree well above 100 and sometimes on the order of several thousand or higher [29,37]. Therefore, an explicit deflation strategy such as forward deflation used by *zroots* and *CO2AFF* is not suitable for solving today's polynomial root problems. Indeed, explicit deflation is at best a semi-stable process since the cumulative effect after many steps may result in a significant loss of accuracy [19]. We tested this notion on the *C02AFF* routine using polynomials with random coefficients; for polynomials of degree higher than 500, the routine would often fail to converge and continue to run for minutes on end. This test serves as the impetus for our research on implicit deflation strategies that help make Laguerre's method a suitable option for solving large degree polynomial equations.

There have been several implicit deflation strategies proposed for Laguerre's method [17,25,33,34]. They differ from the explicit deflation strategies since they result in a modification of Laguerre's method rather than a change to the underlying polynomial. The modifications proposed in [17,34] are mathematically interesting: They have local fourth-order, or higher, convergence and can be run in parallel. However, these methods are not reliable in practice. In [24], it was noted that convergence could be quite slow when two approximations are closer to each other than to a root. This situation frequently arises when dealing with high degree polynomials. Therefore, these methods may only be suitable for iterative refinement. Modifying Laguerre's method using Maehly's procedure [25] is far more practical and has been used to solve matrix and polynomial eigenvalue problems [22,30]. However, this method computes the roots of a polynomial sequentially and, therefore, cannot take advantage of parallel systems.

In contrast, the family of methods derived in [33] allow for the simultaneous approximation of all roots of a polynomial. These methods have local fourth-order convergence and computationally verifiable conditions that guarantee their convergence for simple roots [32]. We are particularly interested in the member of this family that is analogous to Laguerre's method. We will demonstrate that this method has a workload that is well-suited for data-parallelism and use it to develop an algorithm that is effective for solving large degree polynomial equations.

The outline of this article is as follows: In Section 2 we derive an instance of the modified Laguerre method from [33] that we use to form an algorithm for the computation of all roots of a polynomial. In Sections 2.1–2.3, we provide a detailed analysis of the algorithm's initial approximations, stopping criterion, and stability. Finally, in Section 3, the results of several numerical experiments are provided to verify our analysis and the effectiveness of our algorithm.

2 Derivation of the Method

Consider the polynomial p in variable z defined by

$$p(z) = a_0 + a_1 z + \cdots + a_m z^m, \tag{1}$$

where $a_0 a_m \neq 0$. If $a_0 = 0$, then deflate the polynomial and apply the following method to the resulting polynomial. Denote by $\{z_1, \ldots, z_m\}$ the current approximations to the roots $\{\zeta_1, \ldots, \zeta_m\}$ of p. We are interested in updating each approximation z_j until it is "close enough" to the root ζ_j.

Laguerre's method performs this update by solving a quadratic equation, which we denote by $Q_j(z) = 0$. For a derivation of Q_j, see [36]. Note that if the roots of p are real, then both solutions of $Q_j(z) = 0$ are guaranteed to be closer to ζ_j than the current approximation z_j. This is what accounts for the global convergence of Laguerre's method when all roots are real. While global convergence is not guaranteed when the roots are complex, it is our experience that Laguerre's method often performs just as well in this case. A similar sentiment was noted by Parlett in [30].

We denote the roots of Q_j by $\hat{z}_j$ and note that

$$\hat{z}_j = z_j - \frac{m}{G_j \pm \sqrt{(m-1)(mH_j - G_j^2)}}, \tag{2}$$

where

$$G_j = \frac{p^{'}(z_j)}{p(z_j)} = \sum_{i=1}^{m} \frac{1}{(z_j - \zeta_i)} \quad \text{and} \quad H_j = -\left(\frac{p^{'}(z_j)}{p(z_j)}\right)^{'} = \sum_{i=1}^{m} \frac{1}{(z_j - \zeta_i)^2}. \tag{3}$$

The update to z_j is given by the nearest $\hat{z}_j$; that is, we select the sign that maximizes the denominator of the fraction in (2). We reference this update as the *correction term* of z_j.

In [33], two one-parameter families of iteration functions are derived from the classical Weierstrass' method and the Hansen-Patrick formula [16]. For a particular choice of the parameter, a modified Laguerre method is obtained for the simultaneous approximation of all roots of a polynomial. The derivation n [33] allows for repeated roots, but we assume that all roots are simple since we cannot expect to know the multiplicity a priori.

In the case of simple roots, the modified Laguerre method from [33] is equivalent to making the following changes to the terms in (3):

$$G_j = \frac{p^{'}(z_j)}{p(z_j)} - \sum_{\substack{i=1\\ i\neq j}}^{m} \frac{1}{(z_j - z_i)} \quad \text{and} \quad H_j = -\left(\frac{p^{'}(z_j)}{p(z_j)}\right)^{'} - \sum_{\substack{i=1\\ i\neq j}}^{m} \frac{1}{(z_j - z_i)^2}. \tag{4}$$

Moreover, this is equivalent to applying Laguerre's method to the function

$$f_j(z) = \frac{p(z)}{\prod_{\substack{i=1\\ i\neq j}}^{m}(z - z_i)}, \tag{5}$$

when computing the correction term of z_j. Note that this is a classical approach for implementing implicit deflation and it is at the core of the Aberth-Ehrlich-Börsch-Supan method [1,7,11]. Essentially, we are creating poles at all other root approximations and therefore avoiding unnecessary multiple convergences to the same root.

As noted in Section 1, the method from [33] has strong virtues including local fourth-order convergence and a parallel workload. We use this method to form an algorithm for the computation of all roots of a polynomial.

The outline of our algorithm is as follows: We begin with the initial approximations whose computation is discussed in Section 2.1. Then, for a fixed number of iterations, we loop through the array of approximations $(z_1, \ldots, z_m)$. If z_j is not close enough to ζ_j, which is quantified in Section 2.3, then we update the approximation using (2) and (4), as described in Section 2.2. The loop through the array of approximations is considered one iteration of our algorithm. Each iteration can be implemented sequentially or in parallel and both styles are outlined in Algorithm 1 and Algorithm 2, respectively. The former has the benefit of a slightly improved convergence rate, whereas the latter will have the obvious benefit of speedup.

Algorithm 1 Sequential Style

```
$(z_1, \ldots, z_m) \leftarrow$ initial approximations via Algorithm 3
while $i < itmax$ do
    for $j = 1$ to $m$ do
        if $z_j$ is not close enough to $\zeta_j$ then
            Compute roots of $Q_j(\lambda)$ via (2), using (4)
            $z_j \leftarrow$ root that maximizes denominator of (2)
        end if
    end for
    $i \leftarrow i + 1$
end while
```

The **parfor** loops in Algorithm 2 are well-suited for data-parallelism. That is, the parallel tasks are executing the same function across different parts of the dataset. The actual implementation and the effectiveness thereof depends on the appropriate system support, such as a multiprocessor and an

Algorithm 2 Parallel Style

```
$(z_1, \dots, z_m) \leftarrow$ initial approximations via Algorithm 3
while $i < itmax$ do
    parfor $j = 1$ to $m$ do
        if $z_j$ is not close enough to $\zeta_j$ then
            Compute $G_j$ and $H_j$ via (4) and store
        end if
    end parfor
    parfor $j = 1$ to $m$ do
        if $z_j$ is not close enough to $\zeta_j$ then
            Use $G_j$ and $H_j$ to compute roots of $Q_j(\lambda)$ via (2)
            $z_j \leftarrow$ root that maximizes denominator of (2)
        end if
    end parfor
    $i \leftarrow i + 1$
end while
```

operating system that supports multithreading. For now, our focus is on the implementation of the modified Laguerre method from [33] for solving large degree polynomial equations. We consider the parallel implementation a future step in our work, at which point we will consider other features such as adaptivity and multi-precision as is done with MPSolve [6].

2.1 Initial Approximations

As with all iterative methods, the performance of the modified Laguerre method from [33] depends considerably upon the quality of initial approximations. We note that this method has a computationally verifiable condition for the guaranteed convergence to simple roots [32, Theorem 5.2]. However, this does little for us in practice since if the test condition fails we are given no information on how to improve our initial approximations. In this section, we outline a procedure originally due to Bini [4] for the computation of initial approximations that, in practice, almost always converge to the roots of a polynomial.

In essence, this procedure selects complex numbers along circles of suitable radii; what constitutes suitable radii can be formalized by applying Pellet's Theorem [31]. We state the theorem below and note that an elementary proof can be found in [39].

Theorem 1 *Let p be a polynomial as defined in (1). For each k such that $a_k \neq 0$, consider the equation*

$$|a_k|z^k = \sum_{\substack{i=0 \\ i\neq k}}^{m} |a_i|z^i. \tag{6}$$

(i.) If $k = 0$ there exists one real positive solution s_0, and p has no roots of moduli less than s_0.

(ii.) If $0 < k < m$ *there are either no real positive solutions or two real positive solutions* $t_k \leq s_k$*. In the latter case,* p *has no roots in the open annulus* $\mathcal{A}(t_k, s_k)$ *and exactly* k *roots of moduli less than or equal to* t_k*.*

(iii.) If $k = m$ *there exists one real positive solution* t_m*, and* p *has no roots of moduli greater than* t_m*.*

Let $0 = k_1 < k_2 < \cdots < k_q = m$ be the values of k for which (6) has positive solution(s) and let

$$s_0 = s_{k_1} \leq t_{k_2} \leq s_{k_2} \leq \cdots \leq t_{k_{q-1}} \leq s_{k_{q-1}} \leq t_{k_q} = t_m$$

be these solutions. Then, by [4, Theorem 5], any polynomial in the class

$$\mathcal{P}(p) = \left\{ \sum_{i=0}^{m} b_i z^i \ : \ |b_i| = |a_i| \right\}$$

has $(k_{i+1} - k_i)$ zeros in the closed annulus $\bar{\mathcal{A}}(s_{k_i}, t_{k_{i+1}})$, for $i = 1, \ldots, q-1$, and no zeros in the open annulus $\mathcal{A}(t_{k_i}, s_{k_i})$, for $i = 1, \ldots, q$.

While these inclusion results can be used to determine initial approximations for our iterative method, forming positive solutions of (6) requires solving several polynomial equations. Therefore, we seek a strategy with a smaller time complexity. To this end, define

$$\begin{aligned}
u_k &= \max_{i<k} \left| \frac{a_i}{a_k} \right|^{1/(k-i)}, \quad k = 1, \ldots, m, \\
v_k &= \min_{\substack{i>k \\ a_i \neq 0}} \left| \frac{a_i}{a_k} \right|^{1/(k-i)}, \quad k = 0, \ldots, m-1, \\
u_0 &= 1 / \left(1 + \max_{i>0} \left| \frac{a_i}{a_0} \right| \right), \\
v_0 &= 1 + \max_{i<m} \left| \frac{a_i}{a_m} \right|.
\end{aligned} \tag{7}$$

Then it follows from [4, Theorem 4] that the positive solutions of (6) satisfy

$$u_{k_i} \leq t_{k_i} \leq s_{k_i} \leq v_{k_i}, \quad i = 1, \ldots, q. \tag{8}$$

Consider the set $\mathcal{C} = \{(i, \log|a_i|), \ i = 0, 1, \ldots, m\}$ and denote by

$$\gamma(\mathcal{C}) = \{(\hat{k}_i, \log|a_{\hat{k}_i}|), \ 0 = \hat{k}_1 < \hat{k}_2 < \cdots < \hat{k}_{\hat{q}} = m\}$$

the upper envelope of the convex hull of $\mathcal{C}$. By [4, Theorem 6] we know that the vertices of $\gamma(\mathcal{C})$ satisfy $u_l \leq v_l$ if and only if $l \in \{\hat{k}_1, \ldots, \hat{k}_{\hat{q}}\}$. Therefore, by (8), we have $\{k_1, \ldots, k_q\} \subseteq \{\hat{k}_1, \ldots, \hat{k}_{\hat{q}}\}$, where the containing set can be computed efficiently. In fact, a divide and conquer method is used in [4] to compute the convex hull $\gamma(\mathcal{C})$ in $O(m \log m)$ time. Since the set $\mathcal{C}$ is ordered with respect to the first coordinate, we use Andrew's monotone chain algorithm to compute $\gamma(\mathcal{C})$, which, in this case, has time complexity $O(m)$ [2]. The pseudo-code for computing the initial approximations is given in Algorithm 3, where i is the imaginary unit and σ is any nonzero number [4].

Algorithm 3 initial approximations

```
Compute $\hat{k}_1, \ldots, \hat{k}_{\hat{q}}$ via Andrew's monotone chain algorithm.
for $i = 1$ to $\hat{q} - 1$ do
    $n = \hat{k}_{i+1} - \hat{k}_i$
    $u_{\hat{k}_{i+1}} = \left| \frac{a_{\hat{k}_i}}{a_{\hat{k}_{i+1}}} \right|^{1/n}$
    for $j = 1$ to $n$ do
        $z_{\hat{k}_i + j} = u_{\hat{k}_{i+1}} e^{\left(\frac{2\pi}{n} j + \frac{2\pi i}{m} + \sigma\right)\mathrm{i}}$
    end for
end for
```

2.2 Computing the Correction Term

The update of the root approximation z_j requires the computation of the correction term $\hat{z}_j$ via (2), where G_j and H_j are defined in (4). In this section, we outline a numerically stable method for computing the correction term.

Both summations in (4) represent a backward stable computation. Indeed, each summand can be computed in at most 3 flops, each of which is exactly rounded assuming the IEEE 754 standard [14]. Moreover, the summation is well-known to be backward stable [18, Section 4.2]. Finally, polynomial evaluation via the Ruffini-Horner rule is a backward stable computation. We outline the latter result as it will be used when deriving our stopping criterion.

Let $\mathrm{fl}(p(\xi))$ denote the computed value of p evaluated at $\xi \in \mathbb{C}$ using floating-point arithmetic with machine precision μ. Then, if the Ruffini-Horner rule is used, it follows from [4, Theorem 7] that we have

$$\mathrm{fl}(p(\xi)) = \sum_{i=0}^{m} a_i (1 + \epsilon_i) \xi^i, \tag{9}$$

where $|\epsilon_i| < ((2\sqrt{2}+1)i + 1)\mu + O(\mu^2)$. A similar result holds for the first and second derivative of p.

The Ruffini-Horner method is prone to overflow, for instance, when a large degree polynomial with positive coefficients is evaluated at ξ where $|\xi| > 1$. For this reason, we introduce the reversal polynomial p_R in the variable z defined by

$$p_R(z) = a_0 z^m + \cdots + a_{m-1} z + a_m.$$

Let $z \neq 0$ and $\rho = 1/z$, then it is easy to verify that the following holds true

$$\begin{aligned} \frac{p'(z)}{p(z)} &= \rho \left(m - \rho \frac{p_R'(\rho)}{p_R(\rho)} \right), \\ -\left(\frac{p'(z)}{p(z)} \right)' &= \rho^2 \left(m - 2\rho \frac{p_R'(\rho)}{p_R(\rho)} - \rho^2 \left(\frac{p_R'(\rho)}{p_R(\rho)} \right)' \right). \end{aligned} \tag{10}$$

When $|\xi| \leq 1$, we apply the Ruffini-Horner rule to the polynomial p, and its first and second derivative. Otherwise, we apply the Ruffini-Horner rule to the reversal polynomial p_R, and its first and second derivative, evaluated at $\rho = 1/\xi$. In either case, it follows from (10) that we can easily obtain the desired values needed for computing the correction term.

Now that the main components of Algorithm 1 have been established, we analyze our algorithm's time and space complexity. The initial approximations are computed via Algorithm 3 in $O(m)$ time. Then, each approximation is updated in $O(m)$ time using the Ruffini-Horner rule as described above. Assuming that *itmax* is fixed, i.e., does not depend on the degree of the polynomial, it follows that Algorithm 1 can approximate all roots of a polynomial in $O(m^2)$ time. Finally, we note that our algorithm has $O(m)$ space complexity since the required storage is the coefficients of the polynomial, its approximate roots, and the corresponding backward error and condition number.

2.3 Stopping Criterion

Let ξ be an approximation of the root ζ for the polynomial p as defined in (1), where all computations have been done in floating-point arithmetic with machine precision μ. In this section, we derive the backward error of ξ and the condition number of ζ, which are used to establish our stopping criterion and provide a first-order bound on the forward error in our approximation. Both results in Theorem 2 and Theorem 3 can be found in [8]. We provide elementary proofs for completeness and clarity.

For $i = 0, 1, \ldots, m$, denote by e_i an arbitrary representation of the tolerances against which the perturbations Δa_i to a_i will be measured. Then the perturbed polynomial in the variable z is defined by

$$\Delta p(z) = \Delta a_0 + \Delta a_1 z + \cdots + \Delta a_m z^m, \tag{11}$$

and for convenience we define the sign of a complex number z by

$$\operatorname{sgn}(z) = \begin{cases} \bar{z}/|z|, & z \neq 0, \\ 0, & z = 0. \end{cases}$$

The backward error of ξ represents the smallest perturbations Δa_i for which $(p + \Delta p)(\xi) = 0$. Therefore, a natural definition of the backward error of ξ is given by

$$\eta(\xi) = \min\{\epsilon\colon\ (p + \Delta p)(\xi) = 0,\ |\Delta a_i| \leq \epsilon |e_i|,\ i = 0, 1, \ldots, m\}.$$

Note that when $e_i = 1$ we are measuring the absolute backward error and when $e_i = a_i$ we are measuring the relative backward error.

Theorem 2 *The backward error $\eta(\xi)$ is given by*

$$\eta(\xi) = \frac{|p(\xi)|}{\alpha(\xi)}, \tag{12}$$

where $\alpha(\xi) = \sum_{i=0}^{m} |e_i||\xi|^i$.

Proof First we show that the right hand side is a lower bound for $\eta(\xi)$. To this end, note that

$$\begin{aligned}|p(\xi)| &= |\Delta p(\xi)| \\ &\leq \sum_{i=0}^{m} |\xi|^i |\Delta a_i| \leq \epsilon \sum_{i=0}^{m} |\xi|^i |e_i|.\end{aligned}$$

Next, we show that this lower bound is attained by the perturbations

$$\Delta a_i = -\frac{1}{\alpha(\xi)} \operatorname{sgn}(\xi^i) |e_i| p(\xi).$$

To this end, note that

$$\begin{aligned}\Delta p(\xi) &= -\sum_{i=0}^{m} \frac{1}{\alpha(\xi)} \operatorname{sgn}(\xi^i)|e_i| p(\xi) \xi^i \\ &= -\frac{p(\xi)}{\alpha(\xi)} \sum_{i=0}^{m} |e_i||\xi|^i = -p(\xi).\end{aligned}$$

Therefore, $(p + \Delta p)(\xi) = 0$ and $|\Delta a_i|/|e_i| = |p(\xi)|/\alpha(\xi)$ for all $i = 0, 1, \ldots, m$. □

Motivated by the backward error analysis of the Ruffini-Horner rule in (9), we let $e_i = ((2\sqrt{2} + 1)i + 1)a_i$. Then, if the approximate root satisfies

$$\eta(\xi) \leq \mu \tag{13}$$

it follows that ξ is a root of $(p + \Delta p)$, where Δp is defined in (11) and its coefficients Δa_i are no bigger than ϵ_i from (9), for $i = 0, 1, \ldots, m$. Conversely, if $\eta(\xi) > \mu$, then ξ being a root of $(p+\Delta p)$ implies that Δa_i is bigger than ϵ_i, for some $i = 0, 1, \ldots, m$. Therefore, (13) denotes a viable stopping criterion that guarantees iterations do not terminate until $\mathrm{fl}(p(\xi))$ is no longer a reliable computation for updating the approximate root ξ. Furthermore, if ξ is an approximate root that satisfies (13), then the computation of ξ was backward stable. There is no guarantee of backward stability for all polynomials since convergence is not guaranteed. However, in practice, our algorithm almost always performs in a backward stable manner, see Section 3.

As noted in Section 2.2, if $|\xi| > 1$ then we apply the Ruffin-Horner rule to the reversal polynomial p_R at $\rho = 1/\xi$ in order to compute the correction term. In this case, the backward error can be computed as follows

$$\eta(\xi) = \frac{|p_R(\rho)|}{\alpha_R(\rho)}, \tag{14}$$

where $\alpha_R(\rho) = \sum_{i=0}^{m} |e_{m-i}||\rho|^i$. This result is easily verified by noting that

$$\begin{aligned}\frac{|p_R(\rho)|}{\alpha_R(\rho)} &= \frac{|\rho|^m |p(1/\rho)|}{|\rho|^m \alpha(1/\rho)} \\ &= \frac{|p(\xi)|}{\alpha(\xi)}.\end{aligned}$$

The condition number of a root is a measurement of its sensitivity to changes in the coefficients of p. Therefore, a natural definition of the condition number of a nonzero simple root ζ is given by

$$\kappa(\zeta,p)=\limsup_{\epsilon\to 0}\left\{\frac{|\Delta\zeta|}{\epsilon|\zeta|}:\ (p+\Delta p)(\zeta+\Delta\zeta)=0,\ |\Delta a_i|\leq\epsilon|e_i|,\ i=0,1,\ldots,m\right\}.$$

Theorem 3 *The condition number $\kappa(\zeta,p)$ is given by*

$$\kappa(\zeta,p)=\frac{\alpha(\zeta)}{|\zeta||p'(\zeta)|},\tag{15}$$

where $\alpha(\zeta)=\sum_{i=0}^{m}|e_i||\zeta|^i$.

Proof Note the following Taylor series expansions

$$p(\zeta+\Delta\zeta)=p(\zeta)+\Delta\zeta p'(\zeta)+O(\Delta\zeta^2)$$

and

$$\Delta p(\zeta+\Delta\zeta)=\Delta p(\zeta)+\Delta\zeta\Delta p'(\zeta)+O(\Delta\zeta^2).$$

Then the constraint $(p+\Delta p)(\zeta+\Delta\zeta)=0$ implies that

$$\Delta\zeta=-\frac{\Delta p(\zeta)}{p'(\zeta)+\Delta p'(\zeta)}+O(\Delta\zeta^2).$$

Therefore,

$$\frac{|\Delta\zeta|}{\epsilon|\zeta|}\leq\frac{\alpha(\zeta)}{|\zeta||p'(\zeta)+\Delta p'(\zeta)|}+\frac{O(\Delta\zeta^2)}{\epsilon}$$

and by taking the limit as $\epsilon\to 0$ it follows that

$$\kappa(\zeta,p)\leq\frac{\alpha(\zeta)}{|\zeta||p'(\zeta)|}.$$

To show that this upper bound is attained, define $\Delta a_i=-\operatorname{sgn}(\zeta^i)\epsilon|e_i|$. Then we have $-\Delta p(\zeta)=\epsilon\alpha(\zeta)$ and it follows that

$$\frac{|\Delta\zeta|}{\epsilon|\zeta|}=\frac{\alpha\zeta}{|\zeta||(p'(\zeta)+\Delta p'(\zeta)|}+\frac{O(\Delta\zeta^2)}{\epsilon}.$$

Taking the limit as $\epsilon\to 0$ gives the desired equality.

□

Suppose that $\zeta>1$ and define $\rho=1/\zeta$. Then, we compute the condition number as follows

$$\kappa(\zeta,p)=\frac{\alpha_R(\rho)}{|m\cdot p_R(\rho)-\rho\cdot p_R'(\rho)|},\tag{16}$$

where $\alpha_R(\rho) = \sum_{i=0}^{m} |e_{m-i}||\rho|^i$. This result is easily verified by noting that

$$\begin{aligned}\frac{\alpha(\zeta)}{|z||p'(\zeta)|} &= \frac{|\zeta|^m \alpha_R(\rho)}{|\zeta||m\zeta^{m-1}p_R(\rho) - \zeta^{m-2}p_R'(\rho)|} \\ &= \frac{|\zeta|^m \alpha_R(\rho)}{|\zeta|^m |m \cdot p_R(\rho) - \rho \cdot p_R'(\rho)|} = \frac{\alpha_R(\rho)}{|m \cdot p_R(\rho) - \rho \cdot p_R'(\rho)|}.\end{aligned}$$

In summary of the stopping criterion, we compute the backward error $\eta(\xi)$ via (12), if $\xi \leq 1$, or via (14) otherwise. If $\eta(\xi) \leq \mu$, then updates of the root approximation ξ cease and we compute an approximation of the condition number $\kappa(\zeta, p)$. This is done by replacing ζ with the approximate root ξ in (15), if $\xi \leq 1$, or in (16) otherwise. Both values are returned and can be used to determine if the computation of ξ was backward stable and the approximate sensitivity of ζ to changes in the coefficients of p. Finally, the product $\eta(\xi)\kappa(\zeta, p)$ can be used to provide a first-order bound on the forward error in the approximate root ξ.

3 Numerical Experiments

In this section, we present the results of several numerical experiments to verify our analysis in Sections 2.1–2.3 and the effectiveness of Algorithm 1. The results that follow are from tests run on an Intel Core i5 CPU running at 2.7 GHz with 16GB of memory. All code was compiled using *gfortran 8.2.0* with the *-O2* optimization flag in the IEEE double-precision floating-point standard. Fortran 90 source code of Algorithm 1, denoted *FPML*, and all tests is available online at `https://github.com/trcameron/FPML`.

We provide comparisons with two open source Fortran 90 root solvers: *AMVW* from [3] and *Polzeros* from [4]. We are interested in comparing against *AMVW* as it is a recent advancement in eigenvalue methods for computing the roots of a polynomial. It has guaranteed backward stability, assuming the method converges, and quadratic time complexity. Note that we are comparing against the most recent single-shift version of *AMVW* which is maintained at `https://github.com/eiscor/eiscor`.

We are interested in comparing against *Polzeros* as it strongly motivated the development of Algorithm 1 from the initial approximations to the stopping criterion. Furthermore, *Polzeros* is driven by the Aberth-Ehrlich-Börsh-Supan method, which is similar to the modified Laguerre method that we employ. We recognize that *Polzeros* has been superseded by *MPSolve* [5,6] but both make use of the Aberth-Ehrlich-Börsch-Supan method. Furthermore, *MPSolve* is an adaptive multi-precision software written in C, and as of version 3.0, it is implemented in parallel [6]. In the future, we are interested in writing source code for Algorithm 2 in C and applying other features such as adaptivity and multi-precision at which point we will compare against MPSolve. For now, we are comparing against a Fortran 90 version of *Polzeros*, which is available at `https://jblevins.org/mirror/amiller/`.

Table 1 Convergence Rate Tests

Iteration	Error-1	Error-2	Error-3	Error-4	Error-5	Error-6
1	0.71	1.75	1.76	0.83	7.13	3.47
2	0.4	1.27	0.42	0.65	16.6	4.74
3	1.06	0.11	0.84	0.72	0.75	0.75
4	$2.45 \cdot 10^{-4}$	$9.98 \cdot 10^{-2}$	0.4	0.38	0.23	0.82
5	$8.07 \cdot 10^{-15}$	0.94	$9.96 \cdot 10^{-2}$	0.78	0.34	0.16
6	$2.7 \cdot 10^{-17}$	$1.13 \cdot 10^{-9}$	$2.24 \cdot 10^{-4}$	$3.95 \cdot 10^{-2}$	$8.4 \cdot 10^{-4}$	0.52
7	0	$6.84 \cdot 10^{-15}$	$3.41 \cdot 10^{-16}$	$2.96 \cdot 10^{-3}$	$1.08 \cdot 10^{-13}$	$9.68 \cdot 10^{-9}$
8	0	0	0	$2.94 \cdot 10^{-15}$	0	$5.18 \cdot 10^{-14}$
9	0	0	0	0	0	0

We also ran tests on the *C02AFF* routine from the NAG library [28]. For large degree polynomials, the performance of *C02AFF* is not comparable with the other root solvers tested. For this reason, we do not include the results of *C02AFF* in the CPU benchmark tests. However, on small degree polynomials, the *C02AFF* routine performs quite well, so we include the results of *C02AFF* in the accuracy benchmark tests.

In Section 3.1, we demonstrate that the fourth-order convergence of Algorithm 1 can be observed in practice. In Section 3.2, we provide several benchmark tests for comparing the elapsed CPU time of *FPML* with *AMVW* and *Polzeros* when solving an array of polynomial equations. Finally, in Section 3.3, we use accuracy benchmark tests that rely on special polynomials to test for convergence difficulties and the deflation stability of *FPML*, *AMVW*, *Polzeros*, and *C02AFF*.

3.1 Convergence Rate Tests

Six polynomials are used to test the local convergence rate of *FPML*. Each polynomial has random roots in the unit circle of the complex plane. After generating these random roots, we use the *rootstocoeffs* function from the testing in [3] to form the coefficients of the corresponding polynomial. This function relies on multi-precision software for Fortran [27].

Starting with the initial approximations outlined in Section 2.1, the error after each iteration is recorded as the maximum relative forward error in our root approximations. The results are recorded in Table 1, where the column Error-i corresponds to the error in the root approximations for the ith polynomial; the ith polynomial has degree $4 + 4i$, for $i = 0, 1, \ldots, 5$.

Note that fourth-order convergence (or better) can be observed in each column of Table 1, which indicates a surprising feature of the modified Laguerre method from [33]. Often, higher order methods do not display their convergence rate in practice. For instance, the method in [17] has fourth-order convergence, but the approximations must be so close to the roots before this convergence rate will set in that it will rarely be noticed in double-precision floating-point arithmetic.

3.2 CPU Time Benchmark Tests

Our CPU benchmark tests are split into three categories of increasing difficulty: The first category consists of polynomials with random coefficients and the roots of unity. The second category consists of polynomials with well-conditioned roots along a circular curve in the complex plane. The third category consists of a polynomial with random roots in the unit circle and the truncated exponential.

3.2.1 Category One

Each random polynomial has coefficients with real and imaginary parts uniformly distributed on the interval $[-1, 1]$. Figure 1 includes the elapsed time measured in seconds and the error measured as the average maximum backward error over all trials. The backward error is computed via (12), where e_i is defined as in (13). Iterations are run for polynomials of degree 80 to degree 20480, doubling the degree on each step. The number of trials performed ranges from 512 to 2, cutting in half as the degree doubles.

The roots of the polynomial $z^n - 1$ are the n roots of unity: $e^{\frac{2\pi \mathrm{i}}{n} j}$ for $j = 1, \dots, n$. Figure 1 includes the elapsed time measured in seconds and the error measured as the average of the maximum relative forward error over all trials. Again, iterations are run for polynomials of degree 80 to degree 20480, doubling the degree on each step; the number of trials ranges from 512 to 2, cutting in half as the degree doubles.

Note that the backward error of *FPML* and *Polzeros* is nearly identical; similarly, the forward error is the same except for a slight spike from *FPML* at degree 10240. In addition, the elapsed time verifies our time complexity analysis of Algorithm 1, see Section 2.2, since both *AMVW* and *Polzeros* are known to have a quadratic time complexity. Furthermore, the fact that *FPML* is faster than *Polzeros* indicates that the fourth-order convergence of the modified Laguerre method from [33] is making up for its additional cost per iteration over the Aberth-Ehrlich-Börsh-Supan method. Finally, we note that the growth in the backward error of *AMVW* is undesirable as it shows that the resulting approximations ξ are roots of a polynomial whose coefficients are more perturbed than the worst case scenario for $\mathrm{fl}(p(\xi))$, see Section 2.3.

3.2.2 Category Two

Two polynomials are tested: $p_1(z) = \sum_{i=0}^{n} (i+1) z^i$ and $p_2(z) = \sum_{i=0}^{n} \frac{1}{(i+1)} z^i$. Both polynomials have well-conditioned roots that lie on a circular curve in the complex plane. Figure 2 includes the elapsed time measured in seconds and the error measured as the average maximum backward error over all trials. The backward error is computed via (12), where e_i is defined as in (13). Iterations are run for polynomials of degree 80 to degree 10240, doubling the degree on each step. The number of trials performed ranges from 512 to 4, cutting in half as the degree doubles.

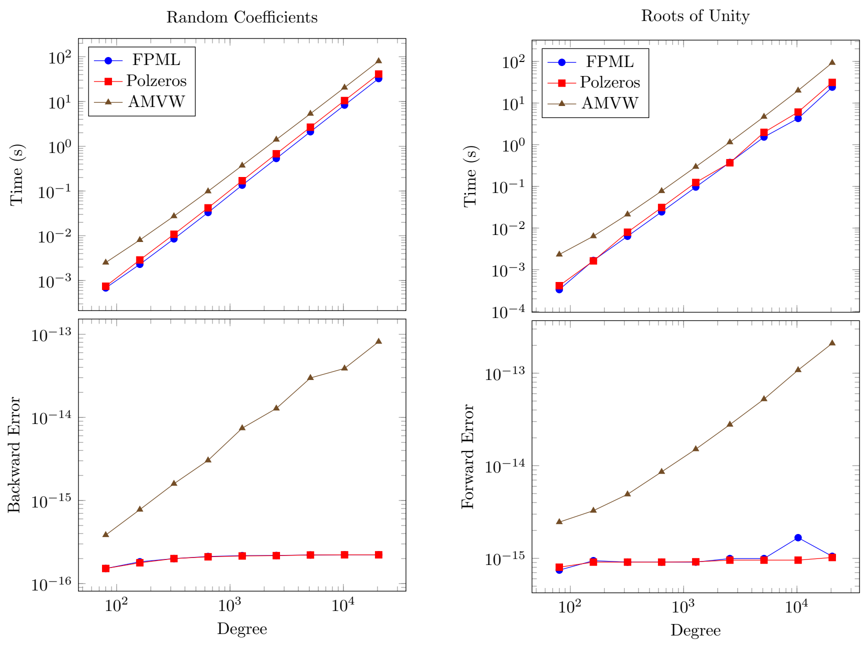

Fig. 1 Category One Testing

Note that the backward error for *FPML*, *Polzeros*, and *AMVW* range between 10^{-16} and 10^{-15} when testing on $p_1(z)$. However, for the testing on $p_2(z)$, there is a significant spike from *Polzeros* that indicates the failure to converge for at least one root approximation. This somewhat surprising failure from *Polzeros* persisted even after increasing the number of iterations.

3.2.3 Category Three

Two polynomials are tested: The first has random roots in the unit circle of the complex plane; these polynomials are created as outlined in Section 3.1. The second is the truncated exponential: $\sum_{i=0}^{m} x^i/i!$. The truncated exponential is well-known to have ill-conditioned roots, especially for larger m. Also, random roots in the unit circle can be ill-conditioned as a result of clustering. In Figure 3, we test on polynomials of degree 10 to degree 100, increasing the degree by 2 on each step. The number of trials is 512 for each degree, and the error is measured as the average maximum backward error over all trials. Again, the backward error is computed via (12), where e_i is defined as in (13).

Note that the backward error of *FPML* and *Polzeros* are both within unit roundoff. However, the backward error of *AMVW* ranges between 10^{-16} and 10^0 as the degree increases from 10 to 100.

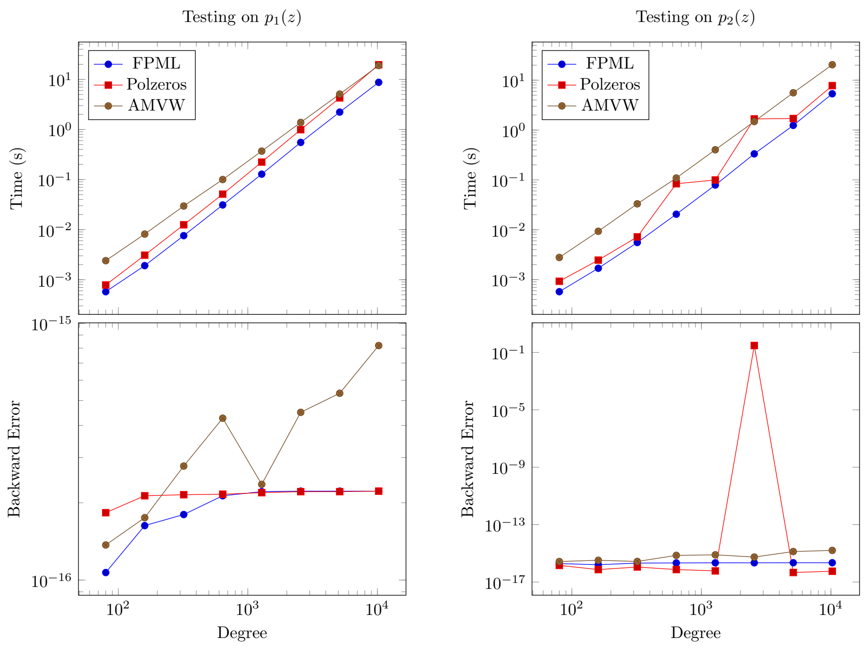

Fig. 2 Category Two Testing

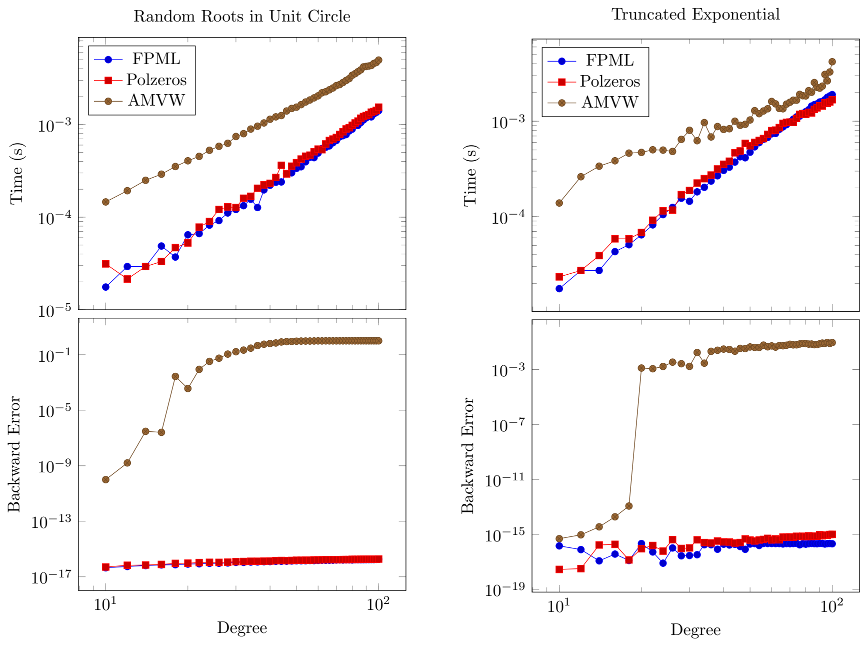

Fig. 3 Category Three Testing

Table 2 Special polynomials tested by Chandrasekaran et al in [9]

Poly No.	Description	Deg.	Roots
1	Wilkinson polynomial	10	$1,\dots,10$
2	Wilkinson polynomial	15	$1,\dots,15$
3	Wilkinson polynomial	20	$1,\dots,20$
4	scale and shifted Wilkinson polynomial	20	$-2.1,-1.9,\dots,1.7$
5	reverse Wilkinson polynomial	10	$1,1/2,\dots,1/10$
6	reverse Wilkinson polynomial	15	$1,1/2,\dots,1/15$
7	reverse Wilkinson polynomial	20	$1,1/2,\dots,1/20$
8	prescribed roots of varying scale	20	$2^{-10},2^{-9},\dots,2^{9}$
9	prescribed roots of varying scale -3	20	$2^{-10}-3,\dots,2^{9}-3$
10	Chebyshev polynomial	20	$\cos(\frac{2j-1}{40}\pi)$
11	$z^{20}+z^{19}+\cdots+z+1$	20	$e^{\mathrm{i}\frac{2j}{21}\pi}$

Table 3 Special polynomials used to test MPSolve in [5]

Poly No.	Description	Deg.	Roots
12	C. Traverso	24	known
13	Mandelbrot	31	known
14	Mandelbrot	63	known

3.3 Accuracy Benchmark Tests

The polynomials in Table 2 are tested in [3,9]. Many of these polynomials are infamous for their ill-conditioned roots and are suitable for testing convergence difficulties, e.g., Wilkinson and Chebyshev polynomial.

Polynomials taken from MPSolve [5] and tested in [3] are given in Table 3. We take the roots computed by MPSolve to be exact. The polynomial provided by Carlo Traverso arises from the symbolic processing of a system of polynomial equations and has multiple roots. The Mandelbrot polynomials are defined recursively as follows: $p_0(z) = 1$ and $p_j(z) = zp_{j-1}(z)^2 + 1$, for $j = 1, 2, \dots, k$, with $m = 2^k - 1$.

The polynomials in Table 4 are from Jenkins and Traub [19] and are designed to test convergence difficulties and the deflation stability of a root finding method. The cubic polynomial $p_1(z) = (z - a)(z + a)(z - 1)$ examines the stopping criterion. Underflow is likely in the polynomial $p_3(z) = \prod_{j=1}^{m}(z-10^{-j})$. Thus, this polynomial may be used to test whether a stopping criterion based on roundoff error fails. To test the performance of our method on multiple or nearly multiple zeros we use the polynomials

$$\begin{aligned}
p_4(z) &= (z-.1)^3(z-.6)(z-.7),\\
p_5(z) &= (z-.1)^4(z-.2)^3(z-.3)^2(z-.4),\\
p_6(z) &= (z-.1)(z-1.001)(z-.998)(z-1.00002)(z-.99999),\\
p_7(z) &= (z-.001)(z-.01)(z-.1)(z-.1+\mathrm{i}a)(z-.1-\mathrm{i}a)(z-1)(z-10),\\
p_8(z) &= (z+1)^5.
\end{aligned}$$

Table 4 Special Polynomials to test root finding algorithms from [19]

Poly No.	Description	Deg.	Roots
15	$p_1(z)$ with $a = 10^{-8}$	3	$a, -a, 1$
16	$p_1(z)$ with $a = 10^{-15}$	3	$a, -a, 1$
17	$p_1(z)$ with $a = 10^{8}$	3	$a, -a, 1$
18	$p_1(z)$ with $a = 10^{15}$	3	$a, -a, 1$
19	$p_3(z)$	10	$10^{-1}, \dots, 10^{-10}$
20	$p_3(z)$	20	$10^{-1}, \dots, 10^{-20}$
21	$p_4(z)$	6	$1/10$ $(m.\ 3), 5/10, 6/10, 7/10$
22	$p_5(z)$	10	$1/10$ $(m.\ 4), 2/10$ $(m.\ 3), 3/10$ $(m.\ 2), 4/10$
23	$p_6(z)$	5	$0.1, 0.998, 1.00002, 0.99999$
24	$p_7(z)$ with $a = 0$	7	$10^{-3}, 10^{-2}, 10^{-1}$ $(m.\ 2), 1, 10$
25	$p_7(z)$ with $a = 10^{-10}$	7	$10^{-3}, 10^{-2}, 10^{-1} - \mathrm{i}a, 10^{-1} + \mathrm{i}a, 1, 10$
26	$p_7(z)$ with $a = 10^{-6}$	7	$10^{-3}, 10^{-2}, 10^{-1} - \mathrm{i}a, 10^{-1} + \mathrm{i}a, 1, 10$
27	$p_8(z)$	5	-1 $(m.\ 5)$
28	$p_9(z)$	20	$10^2 e^{\frac{2\pi \mathrm{i}}{10} j}, 10^{-2} e^{\frac{2\pi \mathrm{i}}{10} j}$
29	$p_{10}(z)$ with $a = 10^3$	3	$a, 1, 1/a$
30	$p_{10}(z)$ with $a = 10^6$	3	$a, 1, 1/a$
31	$p_{10}(z)$ with $a = 10^9$	3	$a, 1, 1/a$
32	$p_{11}(z)$ with $m = 15$	60	$e^{\frac{\pi \mathrm{i}}{2m} j}, 0.9 e^{\frac{\pi \mathrm{i}}{2m} j}$
33	$p_{11}(z)$ with $m = 20$	80	$e^{\frac{\pi \mathrm{i}}{2m} j}, 0.9 e^{\frac{\pi \mathrm{i}}{2m} j}$
34	$p_{11}(z)$ with $m = 25$	100	$e^{\frac{\pi \mathrm{i}}{2m} j}, 0.9 e^{\frac{\pi \mathrm{i}}{2m} j}$

The last convergence test is on the polynomial $p_9(z) = (z^{10} - 10^{-20})(z^{10} + 10^{20})$ whose equimodular roots are known to cause difficulties for algorithms that use powering techniques to separate zeros. Finally, we are interested in testing the deflation stability of our method. To this we end, we use the polynomials

$$\begin{aligned} p_{10}(z) &= (z-a)(z-1)(z-a^{-1}), \\ p_{11}(z) &= \prod_{j=1-m}^{m-1} (z - e^{\frac{\pi \mathrm{i}}{2m} j}) \prod_{j=m}^{3m} (z - .9 e^{\frac{\pi \mathrm{i}}{2m} j}). \end{aligned}$$

The maximum relative forward error from *FPML*, *AMVW*, *Polzeros*, and *C02AFF* when solving for the roots of all polynomials listed in Tables 2–4 is recorded in Table 5. It is clear from Table 5 that *FPML* and Polzeros have comparable accuracy. This is not surprising since the deflation strategies of both these algorithms are similar and the stopping criterion is the same. It is interesting to observe that there are several tests where the performance of *AMVW* is significantly worse than *FPML*, *Polzeros*, and *C02AFF* (e.g., Poly No. $6, 12, 16, 19, 20, 28$). Finally, we note that there are several tests where *C02AFF* is superior to *FPML*, *AMVW*, and *Polzeros* (e.g., Poly No. $1, 2, 3, 23$). The superior performance of *C02AFF* is the effect of a sharper stopping criterion.

Table 5 Accuracy Benchmark Tests

Poly No.	FPML	Polzeros	AMVW	C02AFF
1	$3.04 \cdot 10^{-10}$	$3.38 \cdot 10^{-9}$	$3.12 \cdot 10^{-10}$	$4.25 \cdot 10^{-11}$
2	$3.3 \cdot 10^{-5}$	$5.01 \cdot 10^{-5}$	$6.26 \cdot 10^{-6}$	$4.39 \cdot 10^{-8}$
3	$8.8 \cdot 10^{-2}$	$6.01 \cdot 10^{-2}$	6.48	$4.44 \cdot 10^{-4}$
4	$6.3 \cdot 10^{-11}$	$1.62 \cdot 10^{-12}$	$3.86 \cdot 10^{-12}$	$1.17 \cdot 10^{-12}$
5	$4.44 \cdot 10^{-9}$	$5.29 \cdot 10^{-10}$	$2.16 \cdot 10^{-6}$	$1.03 \cdot 10^{-10}$
6	$2.09 \cdot 10^{-5}$	$2.28 \cdot 10^{-6}$	0.44	$9.88 \cdot 10^{-7}$
7	$8.07 \cdot 10^{-2}$	$6.32 \cdot 10^{-2}$	2.01	$1.97 \cdot 10^{-3}$
8	$4.8 \cdot 10^{-14}$	$5.33 \cdot 10^{-15}$	0.46	$1.15 \cdot 10^{-14}$
9	$6.27 \cdot 10^{-2}$	$6.01 \cdot 10^{-2}$	$8.91 \cdot 10^{-2}$	$2.93 \cdot 10^{-2}$
10	$1.61 \cdot 10^{-9}$	$3.54 \cdot 10^{-10}$	$2.05 \cdot 10^{-10}$	$2.4 \cdot 10^{-11}$
11	$8.28 \cdot 10^{-15}$	$1 \cdot 10^{-15}$	$1.68 \cdot 10^{-15}$	$2.22 \cdot 10^{-15}$
12	$1.73 \cdot 10^{-7}$	$4 \cdot 10^{-7}$	115	$2.9 \cdot 10^{-7}$
13	$1.37 \cdot 10^{-4}$	$9.17 \cdot 10^{-6}$	$4.31 \cdot 10^{-7}$	$1.02 \cdot 10^{-7}$
14	0.29	0.3	0.24	0.68
15	$1.65 \cdot 10^{-16}$	$3.03 \cdot 10^{-19}$	$5.54 \cdot 10^{-9}$	0
16	$1.97 \cdot 10^{-16}$	$1.04 \cdot 10^{-15}$	0.2	$1.97 \cdot 10^{-16}$
17	$3.14 \cdot 10^{-16}$	$2.65 \cdot 10^{-16}$	$5 \cdot 10^{-9}$	$1.49 \cdot 10^{-16}$
18	$2.48 \cdot 10^{-16}$	$2.65 \cdot 10^{-16}$	$3.75 \cdot 10^{-16}$	$1.25 \cdot 10^{-16}$
19	$4.24 \cdot 10^{-16}$	$4.24 \cdot 10^{-16}$	$1.74 \cdot 10^{7}$	$4.24 \cdot 10^{-16}$
20	$4.56 \cdot 10^{-15}$	$3.83 \cdot 10^{-15}$	$3.04 \cdot 10^{16}$	$6.35 \cdot 10^{-16}$
21	$2.86 \cdot 10^{-5}$	$2.68 \cdot 10^{-5}$	$5.98 \cdot 10^{-5}$	$2.05 \cdot 10^{-5}$
22	$1.58 \cdot 10^{-3}$	$1.65 \cdot 10^{-3}$	$1.4 \cdot 10^{-2}$	$1.35 \cdot 10^{-3}$
23	$1.49 \cdot 10^{-4}$	$6.08 \cdot 10^{-5}$	$2.52 \cdot 10^{-5}$	$4.46 \cdot 10^{-7}$
24	$2.82 \cdot 10^{-5}$	$2.92 \cdot 10^{-5}$	$1.41 \cdot 10^{-4}$	$1.6 \cdot 10^{-5}$
25	$2.82 \cdot 10^{-5}$	$2.92 \cdot 10^{-5}$	$1.41 \cdot 10^{-4}$	$1.6 \cdot 10^{-5}$
26	$1.93 \cdot 10^{-5}$	$2.29 \cdot 10^{-5}$	$8.32 \cdot 10^{-5}$	$1.28 \cdot 10^{-5}$
27	$2.35 \cdot 10^{-3}$	$2.11 \cdot 10^{-3}$	$1.19 \cdot 10^{-3}$	0
28	$2.54 \cdot 10^{-15}$	$1.59 \cdot 10^{-16}$	1.01	$3.29 \cdot 10^{-15}$
29	$5.17 \cdot 10^{-23}$	$4.34 \cdot 10^{-22}$	$2.27 \cdot 10^{-16}$	0
30	$9.69 \cdot 10^{-16}$	$1.16 \cdot 10^{-16}$	$2.33 \cdot 10^{-16}$	0
31	$1.38 \cdot 10^{-15}$	$1.19 \cdot 10^{-16}$	$4.14 \cdot 10^{-16}$	0
32	$1.89 \cdot 10^{-7}$	$2.01 \cdot 10^{-7}$	$5.87 \cdot 10^{-8}$	$1.14 \cdot 10^{-7}$
33	$4.94 \cdot 10^{-7}$	$5.42 \cdot 10^{-7}$	$1.98 \cdot 10^{-7}$	$7.23 \cdot 10^{-7}$
34	$7.04 \cdot 10^{-7}$	$4.57 \cdot 10^{-7}$	$2.97 \cdot 10^{-7}$	$1.36 \cdot 10^{-6}$

4 Conclusion

The modified Laguerre method from [33] has strong virtues including local fourth-order convergence and a workload that is well-suited for data-parallelism. The former virtue is proved in [33, Section 4] and we demonstrate the latter in Algorithm 2.

We derived an instance of the modified Laguerre method in 2 and used it to form an algorithm for the approximation of all roots of a polynomial, which we outline in Algorithm 1. Furthermore, we provide a detailed analysis of our algorithm's initial approximations, stopping criterion, and stability in Sections 2.1–2.3.

In Section 3, numerical experiments are provided to verify our analysis and the effectiveness of our algorithm for solving large degree polynomial equations. The experiments in Section 3.1 demonstrate that the fourth-order convergence of Algorithm 1 can be observed in practice. The experiments in Section 3.2 verify the quadratic complexity of our algorithm and highlight its superior speed in comparison to the root solvers *AMVW* from [3] and *Polzeros* from [4]. The experiments in Section 3.3 indicate the reliability of our algorithm even in the face of difficult polynomial equations.

Finally, we note that all computations from FPML in Section 3 were backward stable as the stopping criterion in (13) was satisfied by all root approximations. Future research includes the implementation of Algorithm 2 along with other features such as adaptivity and multi-precision.

Acknowledgements We are grateful to the NAG for providing a free trial that was used in the testing of our work. In addition, we are thankful for numerous private conversations with Dario Bini and David Watkins throughout our research and for the referees' comments and suggestions that significantly improved this article.

References

1. Aberth, O.: Iteration methods for finding all zeros of a polynomial simultaneously. Math. Comp. **27**(122), 339–344 (1973)
2. Andrew, A.M.: Another efficient algorithm for convex hulls in two dimensions. Info. Proc. Letters **9**(15), 216–219 (1979)
3. Aurentz, J.L., Mach, T., Vandebril, R., Watkins, D.S.: Fast and backward stable computation of roots of polynomials. SIAM J. Matrix Anal. Appl. **36**(3), 942–973 (2015)
4. Bini, D.A.: Numerical computation of polynomial zeros by means of Aberth's method. Numer. Algorithms **13**, 179–200 (1996)
5. Bini, D.A., Fiorentino, G.: Design, analysis, and implementation of a multiprecision polynomial root finder. Numer. Algorithms **23**, 127–173 (2000)
6. Bini, D.A., Robol, L.: Solving secular and polynomial equations: A multiprecision algorithm. J. Comput. Appl. Math. **272**, 276–292 (2014)
7. Börsch-Supan, W.: A posteriori error bounds for the zeros of polynomials. Numer. Math. **5**, 380–398 (1963)
8. Chaitin-Chatelin, F., Frayssé, V.: Lectures on Finite Precision Computations. Software, Environments and Tools. SIAM, Philadelphia, PA (1996)
9. Chandrasekaran, S., Gu, M., Xia, J., Zhu, J.: A fast QR algorithm for companion matrices. In: Recent Advances in Matrix and Operator Theory, pp. 111–143. Birkhäuser Basel, Basel, Switzerland (2008)
10. Chen, T.C.: Improving Laguerre's method to cope with symmetry pitfalls in polynomial root-finding. In: Proceedings of the Eigth International Colloquium on Differential Equations, pp. 105–110. VSP, Utrecht, Netherlands (1998)
11. Ehrlich, L.W.: A modified Newton method for polynomials. Commun. ACM **10**(2), 107–108 (1967)
12. Foster, L.V.: Generalizations of Laguerre's method: Higher order methods. SIAM J. Numer. Anal. **18**(6), 1004–1018 (1981)

13. Goedecker, S.: Remarks on algorithms to find roots of polynomials. SIAM J. Sci. Comput. **15**(5), 1059–1063 (1994)
14. Goldberg, D.: What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. **23**, 5–48 (1991)
15. Grad, J., Zakrajšek, E.: LR algorithm with Laguerre shift for symmetric tridiagonal matrices. Comput. J. **15**(3), 268–270 (1972)
16. Hansen, E., Patrick, M.: A family of root finding methods. Numer. Math. **27**, 257–269 (1977)
17. Hansen, E., Patrick, M., Rusnak, J.: Some modifications of Laguerre's method. BIT **17**(4), 409–417 (1977)
18. Higham, N.J.: Accuracy and Stability of Numerical Algorithms. SIAM, Philadelphia, PA (2002)
19. Jenkins, M.A., Traub, J.F.: Principles for testing polynomial zerofinding programs. ACM Trans. Math. Software **1**(1), 26–34 (1975)
20. Kahan, W.: Laguerre's method and a circle which contains at least one zero of a polynomial. SIAM J. Numer. Anal. **4**(3) (1967)
21. Laguerre, E.N.: Oeuvres de Laguerre. Gauthier-Villars and Fils, Paris (1898)
22. Lancaster, P.: Lambda-matrices and Vibrating Systems, *International Series of Monographs on Pure and Applied Mathematics*, vol. 94. Pergamon, Oxford, United Kingdom (1966)
23. Lenard, C.T.: Laguerre's iteration and the method of traces for eigenproblems. Appl. Math. Lett. **3**(3), 73–74 (1990)
24. Leuze, M.R.: A hybrid Laguerre method. BIT **23**(1), 132–138 (1983)
25. Maehly, V.H.: Zur iterativen auflösung algebraisher gleichungen. Z. Angew. Math. Phys. **5**, 260–263 (1954)
26. Mekwi, W.: Iterative Methods for Roots of Polynomials. Master's thesis, University of Oxford, Oxford, England (2001)
27. MPFUN: Multiprecision Software. `http://www.netlib.org/mpfun` (2005). Accessed 18 June 2018
28. NAG: Nag Library, Mark 26. `https://www.nag.com/numeric/fl/nagdoc_latest/html/c02/c02aff.html` (2017). Accessed 27 May 2018
29. Pan, V.Y.: Solving a polynomial equation: some history and recent progress. SIAM Rev. **39**(2), 187–220 (1997)
30. Parlett, B.: Laguerre's method applied to the matrix eigenvalue problem. Math. Comp. **18**(87), 464–485 (1964)
31. Pellet, A.E.: Sur un mode de séparation des racines des équations et la formule de lagrange. Bull. Sci. Math. **5**(2), 393–395 (1881)
32. Petković, M.S., Ilić, S., Rančić, L.: The convergence of a family of parallel zero-finding methods. Comp. Math. Appl. **48**, 455–467 (2004)
33. Petković, M.S., Ilić, S., Tričković, S.: A family of simultaneous zero finding methods. Comput. Math. Appl. **34**(10), 49–59 (1997)
34. Petković, M.S., Petković, L., Živković, D.: Laguerre-like methods for the simultaneous approximation of polynomial zeros. In: Topics in Numerical Analysis, pp. 189–209. Springer Vienna, Vienna (2001)
35. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: Numerical Recipes: The Art of Scientific Computing, 3rd edn. Cambridge University Press, New York, New York (2007)
36. Redish, K.A.: On Laguerre's method. Int. J. of Math. Educ. in Sci. and Tech. **5**(1), 91–102 (1974)
37. Sitton, G.A., Burrus, C.S., Fox, J.W., , Treitel, S.: Factoring very-high-degree polynomials. IEEE Sig. Proc. Mag. **20**, 27–42 (2003)
38. Smith, B.T.: A Zero Finding Algorithm using Laguerre's Method. Master's thesis, University of Toronto, Toronto, Canada (1967)
39. Walsh, J.L.: On Pellet's theorem concerning the roots of a polynomial. Ann. of Math. **26**, 59–64 (1924)
40. Wilkinson, J.H.: Rounding Errors in Algebraic Processes. Prentice-Hall Inc. (1963)