THE LOW-RANK LDL^T QUARTIC SOLVER

PETER STROBACH*

Abstract. A quartic equation $Q(z) = z^4 + Az^3 + Bz^2 + Cz + D$ with *real* coefficients A, B, C, D can be posed as a constrained quadratic form $Q(z) = \mathbf{z}^T\mathbf{Q}(\varphi)\mathbf{z}$, where $\mathbf{z} = [z^2, z, 1]^T$ and $\mathbf{Q}(\varphi)$ is a 3×3 real symmetric matrix with *antidiagonal shift* φ. The φ parameter can be tuned to $\varphi = \varphi_0$ for generating a rank-drop to $rank\{\mathbf{Q}(\varphi_0)\} = 2$. The coefficient vectors $[1, a, b]^T$ and $[1, c, d]^T$ of the quadratic factorization $Q(z) = (z^2 + az + b)(z^2 + cz + d)$ are then easily computed from the rank-2 LDL^T factors of $\mathbf{Q}(\varphi_0)$. This scheme is entirely based on simple and well-conditioned closed-form calculations for the LDL^T factors. The necessary shift parameter φ_0 is computed reliably and accurately as the dominant root of a depressed cubic. We show results of demanding tests. A Fortran subroutine implementation of the new algorithm is released along with the paper.

Key words. Quartic equation, Polynomial root finding, Polynomial factorization, LDL^T factorization, rank reduction.

1. Introduction.

The quartic

$$Q(z) = z^4 + Az^3 + Bz^2 + Cz + D \tag{1.1}$$

is the highest degree polynomial for which a linear factorization

$$Q(z) = (z - z_1)(z - z_2)(z - z_3)(z - z_4) \tag{1.2}$$

can be found by closed-form calculations [1]. The parameters $\{z_1, z_2, z_3, z_4\}$ are also called the roots of $Q(z)$. Applications of quartics typically occur in large nonlinear problems which are reduced to schemes of solving very many small local optimization problems with locally optimal parameters appearing as the roots of a degree-4 polynomial. Other examples can be found in computer graphics [2], for example when computing the intersection of two conic sections [3], or in calculations with ellipses, their intersections and overlap areas [4].

In these applications, it is important that a quartic solver is fast, reliable and sufficiently accurate numerically. The classical closed-form solution by the method of radicals [5], [6] is the fastest, but also the most inaccurate among all possible choices. One can easily find examples in which the classical closed-form quartic solver breaks down. For this reason, the classical closed-form quartic solver is a rather unreliable tool, even when used only as an initial root estimator followed by some refinement, as proposed in [7].

The only way out of this difficulty is a completely new theory for solving quartic equations by quasi closed-form calculations without the drawbacks of the classical solution. This theory is developed in this paper and it is based on a low-rank LDL^T representation of a quartic function. We exploit the fact that there exists a direct relationship between the LDL^T factors of a characteristic matrix that describes the quartic, and the parameters $\{a, b, c, d\}$ of its quadratic factorization

$$Q(z) = (z^2 + az + b)(z^2 + cz + d) \quad . \tag{1.3}$$

The above quadratics are then easily divided in their linear factors and the problem is solved. A special situation occurs in the presence of two complex conjugate factors

*AST-Consulting Inc., Bahnsteig 6, 94133 Röhrnbach, Germany (www.AST-Consulting.net, peter_strobach@gmx.de).

$z_3 = z_1^*$ and $z_4 = z_2^*$. In this case, it is less well-known that besides the real factorization (1.3) there exists the option of factoring the quartic in the complex domain according to:

$$\begin{aligned} Q(z) &= (z - z_1)(z - z_2)(z - z_1^*)(z - z_2^*) \\ &= (z^2 + az + b)(z^2 + a^* z + b^*) \quad , \end{aligned} \tag{1.4}$$

where

$$a = -(z_1 + z_2) \quad , \tag{1.5}$$

$$b = z_1 z_2 \quad . \tag{1.6}$$

In our algorithm, we use both real (1.3) and complex (1.4) type factorizations.

This paper is organized as follows. Section 2 presents an overview of the new low-rank LDL^T quartic solver. Section 3 is devoted to the computation of the shift parameter φ_0 as a central step in this algorithm. Section 4 explains how the LDL^T parameters are practically computed. Section 5 discusses the computation of the quartic roots from the quadratic factors. Section 6 presents a number of interesting experimental results. In Section 7, we establish our conclusions.

2. Algorithm Overview. It is not difficult to verify that the quartic of (1.1) can be posed in terms of the following quadratic form

$$Q(z) = \begin{bmatrix} z^2 & z & 1 \end{bmatrix} \mathbf{Q}(\varphi) \begin{bmatrix} z^2 \\ z \\ 1 \end{bmatrix} \quad , \tag{2.1}$$

where

$$\mathbf{Q}(\varphi) = \begin{bmatrix} 1 & \frac{A}{2} & \left(\frac{B}{6} + \frac{\varphi}{2}\right) \\ \frac{A}{2} & \left(\frac{2B}{3} - \varphi\right) & \frac{C}{2} \\ \left(\frac{B}{6} + \frac{\varphi}{2}\right) & \frac{C}{2} & D \end{bmatrix} \quad . \tag{2.2}$$

The key feature here is that we can introduce a free variable factor φ, also called the *antidiagonal shift* without affecting $Q(z)$ because the powers of z are constant along antidiagonals and the φ cancels out perfectly in the antidiagonal trace.

It is easily seen that the determinant of $\mathbf{Q}(\varphi)$ must be a cubic in φ. Moreover, it turns out that there exists a degree of freedom in "distributing" portions of B along the main antidiagonal in any desired fashion, provided only that these portions of B sum up to the full B along the antidiagonal and that these portions preserve the symmetry of $\mathbf{Q}(\varphi)$. We exploit this additional degree of freedom and construct the special form of $\mathbf{Q}(\varphi)$ as shown in (2.2) so that the corresponding characteristic polynomial $Q(\varphi)$ becomes a *depressed* cubic in φ:

$$Q(\varphi) = \varphi^3 + g\varphi + h \quad , \tag{2.3}$$

with coefficients g and h, where

$$g = AC - 4D - \frac{B^2}{3} \quad , \tag{2.4}$$

$$h = \left(8D + AC - \frac{2B^2}{9}\right)\frac{B}{3} - C^2 - DA^2 \quad . \tag{2.5}$$

Any φ that satisfies $Q(\varphi) = 0$ also satisfies $det\{\mathbf{Q}(\varphi)\} = 0$. In cases where the depressed cubic of (2.3) has three real roots, one of these roots will be especially well suited for our purposes. This is called the *dominant root* φ_0. In Section 3, we give an exact definition of a dominant root and introduce a particularly attractive algorithm for its computation in presence of a *depressed* cubic. This algorithm is a central element in our method and is the deeper reason why we wanted the characteristic cubic precisely in *depressed* form.

For the moment, we can assume that the necessary antidiagonal shift φ_0 has been safely and accurately computed. Then the $\mathbf{Q}(\varphi_0)$ is of rank 2 and we can parametrize this matrix exactly in terms of its rank-2 LDL^T factors as follows:

$$\mathbf{Q}(\varphi_0) = \begin{bmatrix} 1 & \\ \ell_1 & 1 \\ \ell_3 & \ell_2 \end{bmatrix} \begin{bmatrix} 1 & \\ & d_2 \end{bmatrix} \begin{bmatrix} 1 & \ell_1 & \ell_3 \\ & 1 & \ell_2 \end{bmatrix} \quad . \tag{2.6}$$

In simple thinking it is now easy to relate the parameters ℓ_1, ℓ_2, ℓ_3 and d_2 to the elements of $\mathbf{Q}(\varphi_0)$. However, in Section 4 we learn that some deeper considerations must be made for handling roundoff errors in an optimal fashion. For the moment, it is sufficient to assume that these parameters of the low-rank LDL^T representation can be well computed.

Now we can turn to the main issue of constructing the two quadratic factors or subpolynomials of a given quartic from the parameters of this low-rank LDL^T representation. For this purpose, we determine the sign of d_2:

$$\sigma = sign\{d_2\} = \pm 1 \ \text{(either } +1 \text{ or } -1) \quad , \tag{2.7}$$

and introduce a square-root parameter γ as follows:

$$\gamma = \sqrt{\sigma d_2} \quad . \tag{2.8}$$

The characteristic matrix $\mathbf{Q}(\varphi_0)$ of (2.6) can now be posed in the following rank-2 signature standard form:

$$\mathbf{Q}(\varphi_0) = \begin{bmatrix} 1 & \\ \ell_1 & \gamma \\ \ell_3 & \gamma\ell_2 \end{bmatrix} \begin{bmatrix} 1 & \\ & \sigma \end{bmatrix} \begin{bmatrix} 1 & \ell_1 & \ell_3 \\ & \gamma & \gamma\ell_2 \end{bmatrix} \quad . \tag{2.9}$$

Now it is clear that the quartic (1.1) can be expressed as

$$Q(z) = \begin{bmatrix} z^2 & z & 1 \end{bmatrix} \begin{bmatrix} 1 & \\ \ell_1 & \gamma \\ \ell_3 & \gamma\ell_2 \end{bmatrix} \begin{bmatrix} 1 & \\ & \sigma \end{bmatrix} \begin{bmatrix} 1 & \ell_1 & \ell_3 \\ & \gamma & \gamma\ell_2 \end{bmatrix} \begin{bmatrix} z^2 \\ z \\ 1 \end{bmatrix} \quad . \tag{2.10}$$

Apparently, we factor the quartic as we factor its characteristic matrix $\mathbf{Q}(\varphi_0)$. This becomes apparent by introducing the following subpolynomials:

$$p_1(z) = z^2 + \ell_1 z + \ell_3 \quad , \tag{2.11}$$

$$p_2(z) = \gamma z + \gamma\ell_2 \quad , \tag{2.12}$$

Now (2.10) appears in condensed form as follows:

$$Q(z) = [p_1(z),\, p_2(z)] \begin{bmatrix} 1 & \\ & \sigma \end{bmatrix} \begin{bmatrix} p_1(z) \\ p_2(z) \end{bmatrix} = p_1^2(z) + \sigma p_2^2(z) \quad . \tag{2.13}$$

Let z_0 denote any root of $Q(z)$. Then it is clear that

$$Q(z_0) = p_1^2(z_0) + \sigma p_2^2(z_0) = 0 \quad . \tag{2.14}$$

We can now distinguish two cases, depending on the sign parameter σ:

Case 1 ($d_2 \leq 0 : \sigma = -1$): Eq. (2.14) is fulfilled, if either

$$p_1(z_0) = p_2(z_0) \quad , \tag{2.15}$$

or

$$p_1(z_0) = -p_2(z_0) \quad . \tag{2.16}$$

It follows immediately that from (2.16) and using (2.11-12), we can write:

$$q_1(z_0) = p_1(z_0) + p_2(z_0) = z_0^2 + (\ell_1 + \gamma)z_0 + (\ell_3 + \gamma\ell_2) = 0 \quad . \tag{2.17}$$

In the same fashion, (2.15) yields:

$$q_2(z_0) = p_1(z_0) - p_2(z_0) = z_0^2 + (\ell_1 - \gamma)z_0 + (\ell_3 - \gamma\ell_2) = 0 \quad . \tag{2.18}$$

Consequently, the roots of $Q(z)$ are given as the roots of the following two quadratics:

$$q_1(z) = z^2 + (\ell_1 + \gamma)z + (\ell_3 + \gamma\ell_2) \quad , \tag{2.19}$$
$$q_2(z) = z^2 + (\ell_1 - \gamma)z + (\ell_3 - \gamma\ell_2) \quad . \tag{2.20}$$

Case 2 ($d_2 > 0 : \sigma = +1$): Eq. (2.14) is fulfilled, if either

$$p_1(z_0) = i\, p_2(z_0) \quad , \tag{2.21}$$

or

$$p_1(z_0) = -i\, p_2(z_0) \quad , \tag{2.22}$$

where i is the imaginary unit with property $i^2 = -1$. Again, we can easily see that (2.16) yields

$$q_1(z_0) = p_1(z_0) + i\, p_2(z_0) = z_0^2 + (\ell_1 + i\,\gamma)z_0 + (\ell_3 + i\,\gamma\ell_2) = 0 \quad , \tag{2.23}$$

and (2.21) results in

$$q_2(z_0) = p_1(z_0) - i\, p_2(z_0) = z_0^2 + (\ell_1 - i\,\gamma)z_0 + (\ell_3 - i\,\gamma\ell_2) = 0 \quad . \tag{2.24}$$

Consequently, in this case, the $Q(z)$ appears factorized in the form

$$Q(z) = q_1(z)q_2(z) \quad , \tag{2.25}$$

where

$$q_1(z) = z^2 + (\ell_1 + i\,\gamma)z + (\ell_3 + i\,\gamma\ell_2) \quad , \tag{2.26}$$
$$q_2(z) = z^2 + (\ell_1 - i\,\gamma)z + (\ell_3 - i\,\gamma\ell_2) \quad . \tag{2.27}$$

But this is just the special form (1.4) with parameters

$$a = (\ell_1 + i\,\gamma) \quad , \tag{2.28}$$

$$b = (\ell_3 + i\,\gamma\ell_2) \quad , \tag{2.29}$$

according to (1.5-6). Hence it follows immediately that we can compute the desired complex roots z_1 and z_2 as the roots of $q_1(z)$ or, alternatively, their complex conjugates as the roots of $q_2(z)$.

Let us continue working with $q_1(z)$. Using the parameters a and b of (2.28-29), we can express this quadratic in condensed form as follows:

$$q_1(z) = z^2 + az + b \quad . \tag{2.30}$$

Although this quadratic has complex coefficients, we can solve it in an ordinary fashion with

$$z_1 = -\frac{a}{2} + \sqrt{\frac{a^2}{4} - b} \quad , \tag{2.31}$$

$$z_2 = -\frac{a}{2} - \sqrt{\frac{a^2}{4} - b} \quad , \tag{2.32}$$

where the "discriminant" is generally complex here and hence its square-root is the square-root of a complex number. Finally, we should be aware of possible cancellation effects in computing differences of almost identical numbers. For these reasons, we practically select the "large magnitude" solution as follows:

$$\begin{aligned} &if(|z_1| > |z_2|)then \\ &z_{max} = z_1 \\ &else \\ &z_{max} = z_2 \\ &endif \quad . \end{aligned} \tag{2.33}$$

Since $z_1 z_2 = z_{max} z_{min} = b$, we can compute the corresponding "small magnitude" solution with high relative accuracy via

$$z_{min} = \frac{b}{z_{max}} \quad , \tag{2.34}$$

and the algorithm is complete.

3. Computing the Perfect Antidiagonal Shift. The antidiagonal shift φ_0 is a key parameter in the LDL^T algorithm. The final accuracy reached in estimating the quartic roots depends directly on the relative accuracy reached in estimating this shift parameter. We present a geometrical intersection method for computing the φ_0 directly as the dominant root of the depressed cubic (2.3). We also discuss other methods, like eigenvalue based methods which appear reasonable at first glance for solving a problem of this kind.

3.1. The Cubic and the Geometrical Intersection Method. Let us return to the depressed cubic of (2.3). In this Section, we replace φ by x and $Q(\varphi)$ by y for convenience. For any x that satisfies $y = 0$, we can write:

$$x^3 + gx + h = 0 \quad . \tag{3.1}$$

Multiplying both sides of this equation by $1/x$ yields:

$$x^2 + g + \frac{h}{x} = 0 \quad , \tag{3.2}$$

or equivalently

$$x^2 + g = -\frac{h}{x} \quad . \tag{3.3}$$

Apparently, the roots of the depressed cubic are the intersection points of a *parabola*

$$p(x) = x^2 + g \quad , \tag{3.4}$$

and a *reciprocal*

$$r(x) = -\frac{h}{x} \quad . \tag{3.5}$$

Fig. 3.1 (left) illustrates this interpretation for the case $\{g = -1, h = -1\}$. Fig. 3.1 (right) shows the case $\{g = -2, h = -1\}$. The following observations can be made:

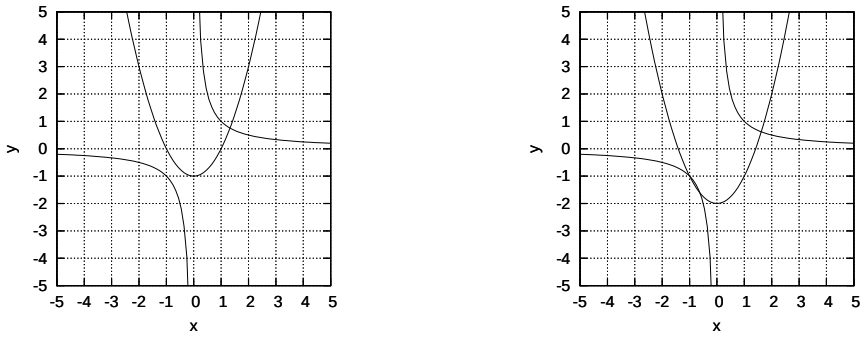

FIG. 3.1. *Intersection of a parabola and a reciprocal for the case $h < 0$. Left: 1 intersection point. Right: 3 intersection points.*

- There are either 1 or 3 intersection points. The case that the curves just touch in the lower-left quadrant (for $h < 0$) corresponds to a real double root of the underlying depressed cubic.
- In any case, one intersection point is *always* located in the upper-right quadrant (for $h < 0$) or in the upper-left quadrant (for $h > 0$). This intersection point constitutes the *dominant root* of the depressed cubic.
- The case $h = 0$ has a trivial solution.
- The dominant root is the intersection point of two *counteracting curves*, i.e., in the entire range of interest (for $h < 0$ the upper-right quadrant), the intersecting curves have gradients of *opposite signs*. This is the reason why the dominant root is always relatively *well conditioned*. In the quadrant of interest, these counteracting curves often intersect under an angle of almost 90 degrees. This becomes apparent from the examples shown in Fig. 3.1. In the second example (right), there are 3 intersection points constituting the 3 real roots of the underlying depressed cubic. But the two real roots in the *opposite* lower-left quadrant are heavily *ill-conditioned* relative to the dominant root in this case. The reason is that these *subdominant roots* are always intersections of two *noncounteracting curves*. These curves have gradients of *equal signs*.

These geometrical facts make it quite attractive to design an algorithm for finding the dominant root of a depressed cubic directly in the intersection space of the parabola and the reciprocal. An algorithm of this kind with guaranteed convergence is next developed.

Begin with a linearization of the reciprocal around a starting point x_0. We have:

$$\overline{r}(x) = r(x_0) + r'(x_0)(x - x_0) \quad , \tag{3.6}$$

where

$$r'(x_0) = \frac{h}{x_0^2} \quad . \tag{3.7}$$

Hence the linearized reciprocal attains the form

$$\overline{r}(x) = -\frac{h}{x_0} + \frac{h}{x_0^2}(x - x_0) \quad . \tag{3.8}$$

This linearized reciprocal intersects in *two* points with the parabola. Consequently, the conditional equation is a *quadratic*, namely

$$x^2 + g = -\frac{h}{x_0} + \frac{h}{x_0^2}(x - x_0) \quad , \tag{3.9}$$

or equivalently:

$$x_0^2 x^2 - hx + (x_0^2 g + 2x_0 h) = 0 \quad . \tag{3.10}$$

By choosing an appropriate initial solution x_0 (we discuss this important issue later), we can always ensure that this linearized reciprocal intersects with the parabola. Hence the above quadratic will *always* have two real roots. Since the linearized reciprocal is always descending, the *minor* (small magnitude) root denoted by x_{min} will in any case be our solution of interest here (independent of the sign of h). So the update is:

$$x_0 \leftarrow x_{min} \quad . \tag{3.11}$$

Our algorithm is a kind of a "bi-iteration" and hence there exists a second step which is required to complete one iteration. This second step consists of a linearization of the parabola around the just found intersection point. The linearization of the parabola is given by

$$\overline{p}(x) = p(x_0) + p'(x_0)(x - x_0) \quad , \tag{3.12}$$

where

$$p'(x_0) = 2x_0 \quad . \tag{3.13}$$

The linearized parabola intersects with the reciprocal. This can be written as

$$x_0^2 + g + 2x_0(x - x_0) = -\frac{h}{x} \quad , \tag{3.14}$$

or equivalently

$$2x_0 x^2 + (g - x_0^2)x + h = 0 \quad . \tag{3.15}$$

Again, a quadratic is obtained. Selecting the root of interest is not quite as simple as before. The curves must intersect because they are *counteracting*. This is a standing principle that holds for all iterations and guarantees convergence of this algorithm in all cases. For the case $h < 0$, these intersections can be found in the upper-right and lower-left quadrants, respectively. In the case $h > 0$, the situation appears just reflected on the vertical axis. Hence we need a case distinction depending on the sign of h to fix our root of interest. Denoting the two real solutions of (3.15) as x_{min} and x_{max}, we can write:

$$x_0 \leftarrow max\{x_{min}, x_{max}\} \quad (h < 0) \quad , \tag{3.16}$$

$$x_0 \leftarrow min\{x_{min}, x_{max}\} \quad (h > 0) \quad , \tag{3.17}$$

and one iteration of the depressed cubic dominant root finder is already complete.

A closer inspection of the geometry of this problem reveals that one can always specify an initial solution x_0 from which the algorithm will converge with a hundred percent probability. For this purpose, we first define a reference point x_r on the reciprocal as the point where the value of the reciprocal just equals the absolute value of its argument. A case distinction is necessary. The corresponding conditional equations for x_r are:

$$r(x_r) = x_r = -\frac{h}{x_r} \quad (h < 0) \quad , \tag{3.18}$$

$$r(-x_r) = x_r = -\frac{h}{-x_r} \quad (h > 0) \quad , \tag{3.19}$$

with solutions:

$$x_r = \sqrt{-h} \quad (h < 0) \quad , \tag{3.20}$$

$$x_r = \sqrt{h} \quad (h > 0) \quad . \tag{3.21}$$

This reference x_r is now used to classify the parameter g in *three regions* with individual rules for the starting point x_0:

Region 1 ($g < -x_r^2$): x_0 is defined as one of the two intersection points of the parabola and the horizontal axis. The conditional equation is:

$$g(x_0) = x_0^2 + g = 0 \quad \rightarrow \quad x_0 = \sqrt{-g} \quad (h < 0) \quad , \tag{3.22}$$

and

$$x_0 = -\sqrt{-g} \quad (h > 0) \quad . \tag{3.23}$$

Region 2 ($-x_r^2 \leq g \leq x_r$): We choose:

$$x_0 = x_r \quad (h < 0) \quad , \tag{3.24}$$

$$x_0 = -x_r \quad (h > 0) \quad . \tag{3.25}$$

Region 3 ($g > x_r$): x_0 is determined so that $r(x_0) = g$. We obtain:

$$r(x_0) = -\frac{h}{x_0} = g \quad \rightarrow \quad x_0 = -\frac{h}{g} \quad . \tag{3.26}$$

These initial values of x_0 ensure that the linearized reciprocal will *always* intersect with the parabola and the iteration is successfully started. Each iteration consists of first computing the linearized reciprocal intersection with the parabola according to (3.10-11) and in a second step, computing the intersection of the linearized parabola with the reciprocal according to (3.15-17). Not more than 4 iterations are typically required for this algorithm to converge. Notice that the method is considerably different from a conventional Newton algorithm working on the cubic directly. Our algorithm has a hundred percent guaranteed convergence when started from one of the above defined initial solutions for x_0. This is an immediate consequence of the fact that we are working with counteracting curves whose intersection points are always unconditionally attracting.

Another aspect is that the underlying expansions in the two steps of our algorithm are *quadratic*, not only linear as known from Newton's iteration. Consequently, we can expect a substantially accelerated convergence. Indeed, our observation is that the method requires not more than 4, sometimes even less than 4 iterations to converge. One of these iterations is typically a very large step, where the algorithm fixes roughly 10 decimal digits in the double precision mantissa, for instance, from an error of $1e-2$ to an error of $1e-12$ in one step. Hence it makes little sense to characterize the rate of convergence of this algorithm in powers like quadratic, cubic or higher, because this would not reflect the typical large step convergence characteristics of this quasi closed-form algorithm. In effect, our algorithm reduces the problem of solving a cubic equation to the elementary problem of solving a sequence of quadratic equations.

Convergence is reached if either the difference between the x_0's obtained from the two subiterations (3.10-11) and (3.15-17) is perfectly zero or drops to a constant value near the machine epsilon as a consequence of finite precision arithmetic. The algorithm terminates on these conditions. Thresholds are not used. Limit cycles near convergence (as known from algorithms based on linearization) were not observed with this algorithm. This features a reliable threshold-free termination of the iterations.

The algorithm determines the dominant root x_0 of the depressed cubic (3.1) or the desired shift φ_0 to high relative accuracy. However, we should be aware that the coefficients g and h of this cubic are not given quantities. They are computed from the quartic coefficients according to (2.4-5). Hence the numerical "bottleneck" in the overall quartic solver is the accuracy of the computed g and h coefficients after (2.4-5). Let us inspect these equations for the limit case of a quartic with a quadruple root at $z = z_q$. The general relation between the coefficients of a quartic and its roots is explicitly given by (recall (1.1-2)):

$$A = -z_1 - z_2 - z_3 - z_4 \quad , \tag{3.27}$$

$$B = z_1 z_2 + (z_1 + z_2)(z_3 + z_4) + z_3 z_4 \quad , \tag{3.28}$$

$$C = -z_1 z_2 (z_3 + z_4) - z_3 z_4 (z_1 + z_2) \quad , \tag{3.29}$$

$$D = z_1 z_2 z_3 z_4 \quad . \tag{3.30}$$

In the special case of a quadruple root $z_1 = z_2 = z_3 = z_4 = z_q$, we obtain:

$$A = -4 z_q \quad , \tag{3.31}$$

$$B = 6 z_q^2 \quad , \tag{3.32}$$

$$C = -4 z_q^3 \quad , \tag{3.33}$$

$$D = z_q^4 \quad . \tag{3.34}$$

We substitute these expressions into (2.4-5). This yields:

$$g = 16z_q^4 - 4z_q^4 - 12z_q^4 = 0 \quad , \tag{3.35}$$

$$h = (8z_q^4 + 16z_q^4 - 8z_q^4)2z_q^2 - 16z_q^6 - 16z_q^6 = 0 \quad . \tag{3.36}$$

A quadruple root is the limit case of a tight root cluster. Hence the above result shows what happens if the quartic roots become increasingly clustered with perhaps a large root mean (i.e. a cluster that appears far from the origin). In such cases, the coefficients g and h will attain very small values in magnitude. In the limit case of a quadruple root, they vanish perfectly. We could demonstrate that the same holds for triple roots with one widely separated root. In essence, we realize that these tiny coefficients in the case of clustered roots with large offset or mean are defined as the differences of large and almost identical numbers. Hence we must be prepared for a drastic loss of significant mantissa digits if the g and h coefficients are computed directly from (2.4-5).

3.2. The Dominant Eigenvalue Approach. A second option exists in computing φ_0 as the dominant (largest magnitude) eigenvalue of a characteristic matrix. Recall that the $\mathbf{Q}(\varphi)$ of (2.2) can be written as

$$\mathbf{Q}(\varphi) = \mathbf{\Phi} - \varphi\mathbf{J} \quad , \tag{3.37}$$

where

$$\mathbf{\Phi} = \begin{bmatrix} 1 & \frac{A}{2} & \frac{B}{6} \\ \frac{A}{2} & \frac{2B}{3} & \frac{C}{2} \\ \frac{B}{6} & \frac{C}{2} & D \end{bmatrix} \quad ; \quad \mathbf{J} = \begin{bmatrix} 0 & 0 & -\frac{1}{2} \\ 0 & 1 & 0 \\ -\frac{1}{2} & 0 & 0 \end{bmatrix} \quad . \tag{3.38}$$

Hence the φ_0 can be computed as the dominant real eigenvalue of the following indefinite generalized symmetric eigenproblem:

$$\mathbf{\Phi v} = \varphi\mathbf{J v} \quad , \tag{3.39}$$

where $\mathbf{v}$ denotes an *eigenvector*. Premultiplying both sides of (3.39) by $\mathbf{J}^{-1}$ where

$$\mathbf{J}^{-1} = \begin{bmatrix} 0 & 0 & -2 \\ 0 & 1 & 0 \\ -2 & 0 & 0 \end{bmatrix} \quad , \tag{3.40}$$

reveals that this problem can also be posed as a conventional nonsymmetric eigenproblem of the form

$$\overline{\mathbf{\Phi}}\mathbf{v} = \varphi\mathbf{v} \quad , \tag{3.41}$$

where

$$\overline{\mathbf{\Phi}} = \mathbf{J}^{-1}\mathbf{\Phi} = \begin{bmatrix} -\frac{B}{3} & -C & -2D \\ \frac{A}{2} & \frac{2B}{3} & \frac{C}{2} \\ -2 & -A & -\frac{B}{3} \end{bmatrix} \quad . \tag{3.42}$$

Both $\mathbf{\Phi}$ and $\overline{\mathbf{\Phi}}$ define their eigenvalues on the level of the original quartic coefficients. Hence there are no cancellation problems on the matrix level. The technique is suitable for an implementation in double precision arithmetic. However, the runtimes can be long and eigenvalue routines may not always be reliable in the expected extreme cases.

3.3. The Third Option. The question appears if there exists yet a third option that pairs the reliability and speed of the depressed cubic and the parabola/reciprocal intersection method with the accuracy of an eigenvalue approach, or even exceeds this accuracy. This option exists and it arises from the observation that the g and h coefficients of the depressed cubic (2.3) are *perfectly invariant* to a linear origin shift of the given quartic (1.1). To see this, introduce a new variable $\overline{z}$ as follows:

$$z = \overline{z} + s \quad , \tag{3.43}$$

where s denotes an arbitrary real *shift*. Substituting z by $\overline{z} + s$ in (1.1) yields the quartic in the new variable $\overline{z}$:

$$Q(\overline{z}) = \overline{z}^4 + \overline{A}\overline{z}^3 + \overline{B}\overline{z}^2 + \overline{C}\overline{z} + \overline{D} \quad , \tag{3.44}$$

where

$$\overline{A} = 4s + A \quad , \tag{3.45}$$

$$\overline{B} = 6s^2 + 3As + B = B + 3s\,(A + 2s) \quad , \tag{3.46}$$

$$\overline{C} = 4s^3 + 3As^2 + 2Bs + C = C + s\,(2B + s\,(3A + 4s)) \quad , \tag{3.47}$$

$$\overline{D} = s^4 + As^3 + Bs^2 + Cs + D = D + s\,(C + s\,(B + s\,(A + s))) \quad . \tag{3.48}$$

We can see that the overlined coefficients are polynomials of growing degrees in s. They can be expressed already in their Horner forms which is the numerically preferable form for evaluating these polynomials for a given shift s. Next we introduce shifted cubic coefficients defined on the coefficients of the shifted quartic as follows:

$$\overline{g} = \overline{A}\,\overline{C} - 4\overline{D} - \frac{\overline{B}^2}{3} \quad , \tag{3.49}$$

$$\overline{h} = \left(8\overline{D} + \overline{A}\,\overline{C} - \frac{2\overline{B}^2}{9}\right)\frac{\overline{B}}{3} - \overline{C}^2 - \overline{D}\,\overline{A}^2 \quad . \tag{3.50}$$

A substitution of (3.45-48) into (3.49) yields:

$$\begin{aligned}
\overline{g} &= 16s^4 + 16As^3 + (8B + 3A^2)s^2 + (4C + 2AB)s + AC \\
&\quad - 4s^4 - 4As^3 - 4Bs^2 - 4Cs - 4D \\
&\quad - 12s^4 - 12As^3 - (4B + 3A^2)s^2 - 2ABs - B^2/3 \\
&= AC - 4D - B^2/3 \\
&= g \quad .
\end{aligned} \tag{3.51}$$

In the same fashion one can demonstrate that

$$\overline{h} = h \quad . \tag{3.52}$$

This reveals that the coefficients of the depressed cubic are *perfectly invariant* to the shift s. For any arbitrary shift, we always satisfy

$$\varphi_0^3 + \overline{g}\varphi_0 + \overline{h} = 0 \quad . \tag{3.53}$$

This amazing property of the shifted depressed cubic is the key to a fast, reliable and accurate computation of φ_0. We can *tune* the shift s in a way so that the

deteriorating cancellation effects in the computation of the $\overline{g}$ and $\overline{h}$ of (3.49-50) are largely minimized.

It turns out that the best strategy to achieve this is to adjust the s so that $\overline{B}$ is absolutely minimized or even perfectly nulled, if possible. In the latter case, we could completely "switch off" an ill-conditioned term in (3.50) and the formulas for $\overline{g}$ and $\overline{h}$ in this case ($\overline{B}=0$) reduce to

$$\overline{g} = \overline{A}\,\overline{C} - 4\overline{D} \quad , \tag{3.54}$$

$$\overline{h} = -\overline{C}^2 - \overline{D}\,\overline{A}^2 \quad . \tag{3.55}$$

The roots of the conditional equation $\overline{B} = 6s^2 + 3As + B = 0$ are formally given by

$$s_{1,2} = -\frac{A}{4} \pm \frac{1}{4}\sqrt{A^2 - \frac{8}{3}B} \quad . \tag{3.56}$$

The primarily desired $\overline{B}=0$ is reached in all cases where the condition

$$3A^2 - 8B > 0 \tag{3.57}$$

is fulfilled. In these cases, we choose the *minor* root s_{min} of the real pair $s_{1,2}$ as shift:

$$s = s_{min} = \frac{-2B}{3A + sign\{A\}\sqrt{9A^2 - 24B}} \;\; ; \quad (3A^2 - 8B > 0) \quad . \tag{3.58}$$

In the special case of a real double root, the shift is simply given by

$$s = -\frac{A}{4} \;\; ; \quad (3A^2 - 8B = 0) \quad . \tag{3.59}$$

We realize that this is a special case known from classical closed form quartic solvers as a "depression". Indeed, this special case of a shift groups the roots of the shifted or "depressed" quartic around the origin so that the *mean* of the shifted roots vanishes perfectly. So in any case where $3A^2 - 8B = 0$, we do not only reach $\overline{B} = 0$, but simultaneously also satisfy $\overline{A} = 4s + A = 0$ according to (3.45).

Finally, it remains to consider the case of a complex conjugate root pair $s_{1,2}$. We are not interested in complex shifts here. Hence in this case, we also set

$$s = -\frac{A}{4} \;\; ; \quad (3A^2 - 8B < 0) \quad , \tag{3.60}$$

and hereby fix the shift to the *real part* of the complex conjugate pair $s_{1,2}$. What is the effect of this choice on the $\overline{B}$? To see this, consider

$$\frac{\partial \overline{B}}{\partial s} = 12s + 3A \quad . \tag{3.61}$$

Apparently, a shift $s = -A/4$ determines the $\overline{B}$ so that $\partial\overline{B}/\partial s = 0$. An extremal point of the function $\overline{B}(s)$ is reached. This extremal point can only be an *absolute minimum* and is unique, because $\overline{B}(s)$ is either a pot or an upside-down pot and does not cross the horizontal axis.

Our shift strategy either pins $\overline{B}$ to zero or pins $\overline{A}$ to zero or both. If $\overline{A}$ is pinned to zero then $\overline{B}$ must attain an absolute minimum. The effect of this "B-shifting" is that in the overlined domain, we "switch off" or largely attenuate a large

term in the h-formula and overall subtract "less equal" numbers in the overlined domain than in the original domain because we remove much of the mean that exists in the problem by shifting. However, a *general* elimination of the root-mean by a conventional "depression" shift $\overline{A} = 0$ is not a clever choice here because the $\overline{A}$ does not appear as a "switch" in our formula for the critical $\overline{h}$ and therefore, an $\overline{A} = 0$ is much less valuable for our purposes than a $\overline{B} = 0$. For a $\overline{B} = min$, we get the $\overline{A} = 0$ anyway as a by-product "for free". Finally, notice that the Horner forms of (3.46-48) play an important role in this concept because they allow us to compute the overlined B, C and D coefficients with high relative accuracy.

Let us conclude this Section by looking at the B-shift in the case of a perfect quadruple root. From (3.31-32), we know that in this case, we have $A = -4z_q$ and $B = 6z_q^2$. We can immediately verify that in this case, we obtain $3A^2 - 8B = 0$ which is just the special case (3.59). Hence B-shifting results in both $\overline{B} = 0$ and $\overline{A} = 0$. This is a case of depression and the quadruple root is moved into the origin of the overlined domain. Consequently, we also have $\overline{C} = 0$ and $\overline{D} = 0$. This in turn forces the overlined g and h coefficients to zero and the result is a perfect $\varphi_0 = 0$. A double precision routine for computing the φ_0 along this B-shifting concept will produce this $\varphi_0 = 0$ exactly in this case and the same holds for triple roots. Hence arbitrarily higher precision arithmetic would not result in any improvement here anymore. The errors that still may occur result from the rest of the algorithm which is very accurately implemented, as we shall see in the next Section. Hence even perfect results do not come as a surprise as soon as one knows how this algorithm works. Outstanding or even perfect results can be expected in all cases of triple and quadruple roots. These cases are a simple exercise for the low-rank LDL^T quartic solver while they appear as the most demanding cases for many other algorithms. It comes without saying that in applications where only triple or quadruple roots can occur, we can fix the φ_0 directly to zero without computing it explicitly.

4. Fitting the LDL^T Parameters. Once the φ_0 is known, we can fit the LDL^T model of (2.6) onto the characteristic matrix of (2.2). This can be expressed in a layerwise fashion as follows:

$$\begin{bmatrix} 1 & \frac{A}{2} & \left(\frac{B}{6}+\frac{\varphi_0}{2}\right) \\ \frac{A}{2} & \left(\frac{2B}{3}-\varphi_0\right) & \frac{C}{2} \\ \left(\frac{B}{6}+\frac{\varphi_0}{2}\right) & \frac{C}{2} & D \end{bmatrix} = \begin{bmatrix} 1 & \ell_1 & \ell_3 \\ \ell_1 & \ell_1^2 & \ell_1\ell_3 \\ \ell_3 & \ell_1\ell_3 & \ell_3^2 \end{bmatrix} + d_2 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & \ell_2 \\ 0 & \ell_2 & \ell_2^2 \end{bmatrix} . \tag{4.1}$$

The first layer is completely determined by

$$\ell_1 = \frac{A}{2} \quad , \tag{4.2}$$

$$\ell_3 = \frac{B}{6} + \frac{\varphi_0}{2} \quad . \tag{4.3}$$

The second layer reduces to a 2×2 symmetric fitting problem because in the ideal case (without considering roundoff error), we satisfy

$$\begin{bmatrix} \left(\frac{2B}{3}-\varphi_0\right) & \frac{C}{2} \\ \frac{C}{2} & D \end{bmatrix} - \begin{bmatrix} \ell_1^2 & \ell_1\ell_3 \\ \ell_1\ell_3 & \ell_3^2 \end{bmatrix} = d_2 \begin{bmatrix} 1 & \ell_2 \\ \ell_2 & \ell_2^2 \end{bmatrix} \quad , \tag{4.4}$$

or equivalently (by elimination of φ_0)

$$\begin{bmatrix} (B-2\ell_3-\ell_1^2) & (\frac{C}{2}-\ell_1\ell_3) \\ (\frac{C}{2}-\ell_1\ell_3) & (D-\ell_3^2) \end{bmatrix} = d_2 \begin{bmatrix} 1 & \ell_2 \\ \ell_2 & \ell_2^2 \end{bmatrix} \quad . \tag{4.5}$$

This constitutes a set of relations for the d_2 and ℓ_2 parameters. A naive solution is:

$$d_2 = B - 2\ell_3 - \ell_1^2 \quad , \tag{4.6}$$

$$\ell_2 = \left(\frac{C}{2} - \ell_1\ell_3\right)/d_2 \quad , \tag{4.7}$$

This choice leaves the bottom-right relation in (4.5), namely

$$D - \ell_3^2 = d_2\ell_2^2 \quad , \tag{4.8}$$

completely unattended. This equation holds only if ideal infinite precision arithmetic is assumed. In finite precision arithmetic, this equation will no longer hold and in cases of nearly triple or quadruple root like clusters, we can expect to be faced with extreme errors $\epsilon = D - \ell_3^2 - d_2\ell_2^2$, if d_2 and ℓ_2 are computed according to (4.6-7).

To cope with this difficulty, we must exploit the fact that the problem is *overdetermined*. Two parameters d_2 and ℓ_2 are determined by 3 equations. There exists a degree of freedom that can be used for making the implementation more resistant against cancellation errors in forming differences of almost identical numbers. For this purpose, we rewrite (4.5) in vector form as follows:

$$\begin{bmatrix} B-2\ell_3-\ell_1^2 \\ \frac{C}{2}-\ell_1\ell_3 \\ D-\ell_3^2 \end{bmatrix} = d_2\ell_2 \begin{bmatrix} 1/\ell_2 \\ 1 \\ \ell_2 \end{bmatrix} \quad . \tag{4.9}$$

We can introduce an intermediate quantity

$$\delta_2 = 2d_2\ell_2 = C - A\ell_3 \quad , \tag{4.10}$$

to see that there are now two options left for calculating ℓ_2 either from the top row of (4.6) according to

$$\ell_2^{(1)} = \frac{\delta_2}{2(B-2\ell_3-\ell_1^2)} \quad , \tag{4.11}$$

or from the bottom row of (4.6) according to

$$\ell_2^{(2)} = \frac{2(D-\ell_3^2)}{\delta_2} \quad . \tag{4.12}$$

In infinite precision arithmetic, we have $\ell_2^{(1)} = \ell_2^{(2)}$. In finite precision arithmetic, these two forms of ℓ_2 may assume significantly different values as a consequence of cancellation errors in computing the terms $C - A\ell_3$ of (4.10), $B - 2\ell_3 - \ell_1^2$ in (4.11) and $D - \ell_3^2$ in (4.12).

There exists no way to bypass the cancellation error in the computation of the $\delta_2 = C - A\ell_3$ according to (4.10) because this quantity is used in both forms of ℓ_2 according to (4.11-12). However, it makes sense to study the cancellation errors of

the terms $B - 2\ell_3 - \ell_1^2$ in (4.11) and $D - \ell_3^2$ in (4.12). These cancellation errors are most severe in the case of clustered roots. Hence we evaluate these expressions in the limit case of a quadruple root at $z = z_q$ and obtain (using (3.31-34) and $\varphi_0 = 0$ from (3.35-36)):

$$B - 2\ell_3 - \ell_1^2 = 4z_q^2 - 4z_q^2 = 0 \quad , \tag{4.13}$$

$$D - \ell_3^2 = z_q^4 - z_q^4 = 0 \quad . \tag{4.14}$$

We observe that the cancellation errors in (4.14) appear as differences of z_q^4 while the cancellation errors in (4.13) appear only as differences of z_q^2. Cancellation errors in differences of z_q^4 are much more severe than cancellation errors in differences of z_q^2. Hence the top requirement here is to minimize these higher powers of z_q cancellation errors.

This is achieved if we choose $\ell_2 = \ell_2^{(1)}$ in cases where $D > 0$ and choose $\ell_2 = \ell_2^{(2)}$ otherwise. Exceptions of $B - 2\ell_3 - \ell_1^2 = 0$ in (4.11) or $\delta_2 = 0$ in (4.12) must be handled by setting $\ell_2 = 0$. Finally, we compute the d_2 via

$$d_2 = \frac{\delta_2}{2\ell_2} \quad , \tag{4.15}$$

provided $\ell_2 \neq 0$. Otherwise, we set $d_2 = 2B/3 - \varphi_0 - \ell_1^2$ to satisfy (4.1). A case with $\ell_2 = 0$ and $d_2 = 0$ occurs in the presence of an ideal quadruple root, where $\mathbf{Q}(\varphi_0 = 0)$ drops to a rank of one.

4.1. Handling Bi-Quadratic Equations. An exceptional case is the so-called bi-quadratic equation

$$Q(z) = z^4 + Bz^2 + D \quad . \tag{4.16}$$

The bi-quadratic equation is just a restricted quartic with coefficients $A = 0$ and $C = 0$. A bi-quadratic equation is easily solved as a quadratic equation followed by a square-rooting of the roots. In our general quartic solver, we must handle the bi-quadratic equation as an exception because the δ_2 vanishes perfectly in this case (recall (4.10)) as a consequence of the vanishing coefficients A and C. A remedy exists in computing d_2 using (4.6), where we omit the ℓ_1 because this term vanishes as well as a consequence of the vanishing A. Consequently, if both $A = 0$ and $C = 0$ are detected, we compute

$$d_2 = B - 2\ell_3 \quad , \tag{4.17}$$

and leave the ℓ_2 fixed to zero because this quantity vanishes in the bi-quadratic case, as we can easily verify from (4.7).

5. Solving the Quadratics. A final issue is the computation of the roots of the quartic as the roots of the two quadratic factors. In *Case 2* ($\sigma = +1$) this is completely handled already in (2.28-34). In *Case 1* ($\sigma = -1$), however, we must compute the roots of the two quadratic factors

$$q_1(z) = z^2 + az + b \quad , \tag{5.1}$$

$$q_2(z) = z^2 + cz + d \quad , \tag{5.2}$$

where (recall (2.19-20)):

$$a = \ell_1 + \gamma \quad , \tag{5.3}$$

$$c = \ell_1 - \gamma \quad , \tag{5.4}$$

$$b = \ell_3 + \gamma\ell_2 \quad , \tag{5.5}$$

$$d = \ell_3 - \gamma\ell_2 \quad . \tag{5.6}$$

These coefficients of the quadratics (5.1-2) cannot be computed "just like that" because there exists an obvious risk of running into unacceptable cancellation errors. For instance, if ℓ_1 and γ are equally signed, then a is accurate while c will suffer from a cancellation of mantissa digits by subtraction and is hence the less "trustworthy" quantity. The same situation occurs for the coefficients b and d. In essence, two of these coefficients are "trustworthy" while the other two's must be replaced by some computation via a different path circumventing the subtraction of equally signed numbers.

This alternative path can be constructed via a back-connection of the quadratic coefficients with the original quartic coefficients by convolution. We can write:

$$\begin{bmatrix} A-a \\ B-b \\ C \\ D \end{bmatrix} = \begin{bmatrix} 1 & \\ a & 1 \\ b & a \\ & b \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix} \quad . \tag{5.7}$$

From the last row in (5.7), we can immediately see that

$$D = bd \quad . \tag{5.8}$$

This is the required second path for circumventing cancellation errors in the computation of the coefficient pair b and d. We compare the magnitudes of b and d. The larger magnitude quantity is always the trustworthy quantity. The smaller magnitude quantity is computed via the second path (5.8). This yields the following post-processing scheme for the coefficients b and d:

$$d = \frac{D}{b} \quad (|d| < |b|) \quad , \tag{5.9}$$

$$b = \frac{D}{d} \quad otherwise \quad . \tag{5.10}$$

Now as we fixed the b and d coefficients, we find that essentially the same situation exists in case of the a and c coefficient pair. Again, we exploit the fact that these coefficients are linked via the original quartic coefficients. Excluding the bottom row of (5.7), we can write:

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \begin{bmatrix} a+c \\ b+ac+d \\ bc+ad \end{bmatrix} \quad . \tag{5.11}$$

In the event that $|a| > |c|$, we can solve this system for c

$$\begin{bmatrix} A-a \\ B-b-d \\ C-ad \end{bmatrix} = \begin{bmatrix} 1 \\ a \\ b \end{bmatrix} c \;\rightarrow\; c \quad , \tag{5.12}$$

which is now easily recognized as an overdetermined system of 3 linear equations for the parameter c. We solve in a standard fashion in the least-squares sense for the desired parameter c.

On the other hand, if a is the smaller magnitude and hence the "untrustworthy" quantity, we can compute this quantity alternatively as the solution of the following fitting problem in a standard least-squares sense:

$$\left[\begin{array}{c} A - c \\ B - b - d \\ C - bc \end{array} \right] = \left[\begin{array}{c} 1 \\ c \\ d \end{array} \right] a \quad \rightarrow \quad a \quad . \tag{5.13}$$

Besides these links for the "cross-quadratic" $\{b, d\}$ and $\{a, c\}$ coefficient pairs, there exists a second link between the "auto-quadratic" pairs $\{a, b\}$ and $\{c, d\}$ and the given quartic coefficients. To see this, notice that the top and bottom rows of (5.7) can be used to express the $\{c, d\}$ pair in terms of the $\{a, b\}$ pair as follows:

$$c = A - a \quad , \tag{5.14}$$

$$d = D/b \quad . \tag{5.15}$$

Substituting these expressions into the two center rows of (5.7) yields:

$$(B - b)b = ab(A - a) + D \quad , \tag{5.16}$$

$$bC = b^2(A - a) + aD \quad . \tag{5.17}$$

An error vector is established:

$$\mathbf{e} = \left[\begin{array}{c} (B - b)b + ab(a - A) - D \\ bC + b^2(a - A) - aD \end{array} \right] \quad . \tag{5.18}$$

A Jacobian is established:

$$\mathbf{F} = \left[\begin{array}{cc} \frac{\partial \mathbf{e}}{\partial a} & \frac{\partial \mathbf{e}}{\partial b} \end{array} \right] = \left[\begin{array}{cc} (2ab - bA) & (B - 2b + a^2 - aA) \\ (b^2 - D) & (C + 2ba - 2bA) \end{array} \right] \quad . \tag{5.19}$$

Updates for the parameters a and b are computed via solving the following 2×2 system of linear equations:

$$\mathbf{F} \left[\begin{array}{c} \Delta a \\ \Delta b \end{array} \right] = -\mathbf{e} \quad \rightarrow \quad \Delta a, \, \Delta b \quad , \tag{5.20}$$

and the pair $\{a, b\}$ is eventually updated via

$$a \leftarrow a + \Delta a \quad , \tag{5.21}$$

$$b \leftarrow b + \Delta b \quad . \tag{5.22}$$

These updates will not necessarily improve the result. Therefore, after each update, a test is performed whether or not the updated coefficients reduce the absolute coefficient error

$$E = |e_1| + |e_2| \quad , \tag{5.23}$$

where $\mathbf{e} = [e_1, \, e_2]^T$ and $\mathbf{e}$ is the error vector of (5.18). In most of these cases, this test fails and we restore the "old" coefficients prior to updating meaning that the routine

is practically *inactive*. However, in cases in which this test is passed, the coefficient refinement can be significant. Several iterations of this kind are eventually performed until either the overall absolute coefficient error E of (5.23) is perfectly nulled or the E of the actual update is greater than the E of the previous update. Then the routine is stopped and the previous parameters are restored. Hence this routine operates like a final "watchdog" that observes whether or not a final improvement is possible by linking the quadratic constituted by the $\{a, b\}$ coefficient pair directly with the coefficients of the quartic in a final backward correction step. The same principle is applied prior to solving the quadratic constituted by the $\{c, d\}$ pair.

6. Experimental Verification. Results of computer simulations are shown here for the proposed low-rank LDL^T quartic solver and two reference methods, namely: (1) The Companion matrix eigenvalue method using Lapack subroutine *dgeev.f* [9] working on a quartic coefficient Companion matrix of dimension 4×4 and (2): The POLZEROS root-finder of [10] (a Fortran subroutine of the method can be found under [11]).

Single trial runs are not sufficiently informative regarding the robustness and performance of a root finder. Therefore we also show results of random tests with many trial runs of statistically independent realizations of quartics with special properties, for instance quartics with large root spread or quartics with tightly clustered roots with large offset. The random component in all these tests is generated using the `random_number` generator in Fortran. Repeated calls on this function generate sequences $\{\rho_k\}$ of random numbers uniformly distributed in the interval $[0 : 1]$.

6.1. Tests Using Random Coefficient Quartics. A standard but largely uncritical test for polynomial root-finder algorithms are random coefficient polynomials. We begin with a test of this kind for orientation purposes. The quartic coefficient A is a random number generated according to

$$A = 2(\rho_k - 0.5) \quad , \tag{6.1}$$

where k is the realization index. The B, C and D coefficients are constructed in the same fashion. The test comprises 25000 statistically independent realizations or trial runs using the algorithms under comparison for identifying the roots of these random quartics. As a reference, we use "true" or "ideal" roots from a root-finder operating in an extended precision environment.

The criterion monitored in each trial run is the *absolute root error* defined as

$$e(k) = |z(k) - \hat{z}(k)| \,; \;\; k = 1, 2, 3, \ldots, 10^5 \quad , \tag{6.2}$$

where $z(k)$ is a true root and $\hat{z}(k)$ is the corresponding estimated root. Each of the 25000 trials in a random test generates 4 root errors. Consequently we generate, in each experiment, an error sequence consisting of 10^5 absolute root error samples.

The $\{e(k)\}$ sequence is a strictly positive random process and is hence characterized by a *probability density function* (PDF) $p(e)$. The probability that an absolute root error sample occurs in the range $[0 \leq e \leq \infty]$ is exactly 1 and is formally given as follows:

$$P(0 : \infty) = \int_{e=0}^{\infty} p(e) de = 1 \quad . \tag{6.3}$$

Our criterion is directly related to this elementary statistical fact and is given by the

probability that an absolute root error sample exceeds a reference error e_{ref}:

$$P(e > e_{ref}) = P(e_{ref} : \infty) = \int_{e=e_{ref}}^{\infty} p(e)de \leq 1 \quad . \tag{6.4}$$

In estimation theory, this criterion is often named a *receiver operating characteristics* (ROC) [12]. The ROC is the most instructive and informative criterion for evaluating the performance of an estimator. A root-finder is an estimator. Hence the ROC is the right criterion for performance evaluation here.

Fig. 6.1 shows the mean absolute root error sequences in this experiment for the Companion-eigenvalue quartic solver using Lapack dgeev.f (Top image in Fig. 6.1), the POLZEROS root finder (Center image in Fig. 6.1), and finally the low-rank LDL^T quartic solver (Bottom image in Fig. 6.1). Fig. 6.2 shows the statistical evaluation of these errors in terms of the ROC-curves. We find that POLZEROS performs slightly better than LDLT. Both LDLT and POLZEROS perform by approximately one order of magnitude or one decimal digit better than DGEEV.

6.2. Tests Using Random Quartics with Tight Root Clusters. A much more critical test series is now shown using quartics with tight root clusters and large root means or offsets. Cases of multiple roots can be handled as well (See Section 6.5 for examples). Special algorithms exist for handling multiple roots [13].

6.2.1. Clustered Real Roots. The root model used in this experiment is given as follows:

$$z_i(k) = 1000 + \rho_{i,k}\,, \;\; i = 1, 2, 3, 4\,, \;\; k = 1, 2, 3, \ldots, 25000 \quad , \tag{6.5}$$

where $\rho_{i,k}$ denotes the random component of root i uniformly distributed in the range $[0 : 1]$ as usual. Fig. 6.3 shows the resulting root errors already evaluated in terms of their ROC curves obtained for the 3 algorithms under test. It can be seen that POLZEROS and DGEEV show a similar characteristics. LDLT performs clearly better.

6.2.2. Clustered Complex Conjugate Root Pairs. The root model used in this experiment is given by

$$z_1(k) = 600 + (\rho_{1,k} - 0.5) + i(\rho_{2,k} - 0.5) \quad , \tag{6.6}$$
$$z_1^*(k) = 600 + (\rho_{1,k} - 0.5) - i(\rho_{2,k} - 0.5) \quad , \tag{6.7}$$
$$z_2(k) = 600 + (\rho_{3,k} - 0.5) + i(\rho_{4,k} - 0.5) \quad , \tag{6.8}$$
$$z_2^*(k) = 600 + (\rho_{3,k} - 0.5) - i(\rho_{4,k} - 0.5) \quad . \tag{6.9}$$

Fig. 6.4 shows the ROC-curves for this case. Again, LDLT comes up with the top performance. POLZEROS approaches its breakdown region.

6.3. Random Quartics with Large Root Spread. In this experiment, the root model is given as follows:

$$z_1(k) = 1 + 0.1\rho_{1,k} \quad , \tag{6.10}$$
$$z_2(k) = 10^3 + 10^2\rho_{2,k} \quad , \tag{6.11}$$
$$z_3(k) = 10^6 + 10^5\rho_{3,k} \quad , \tag{6.12}$$
$$z_4(k) = 10^9 + 10^8\rho_{4,k} \quad . \tag{6.13}$$

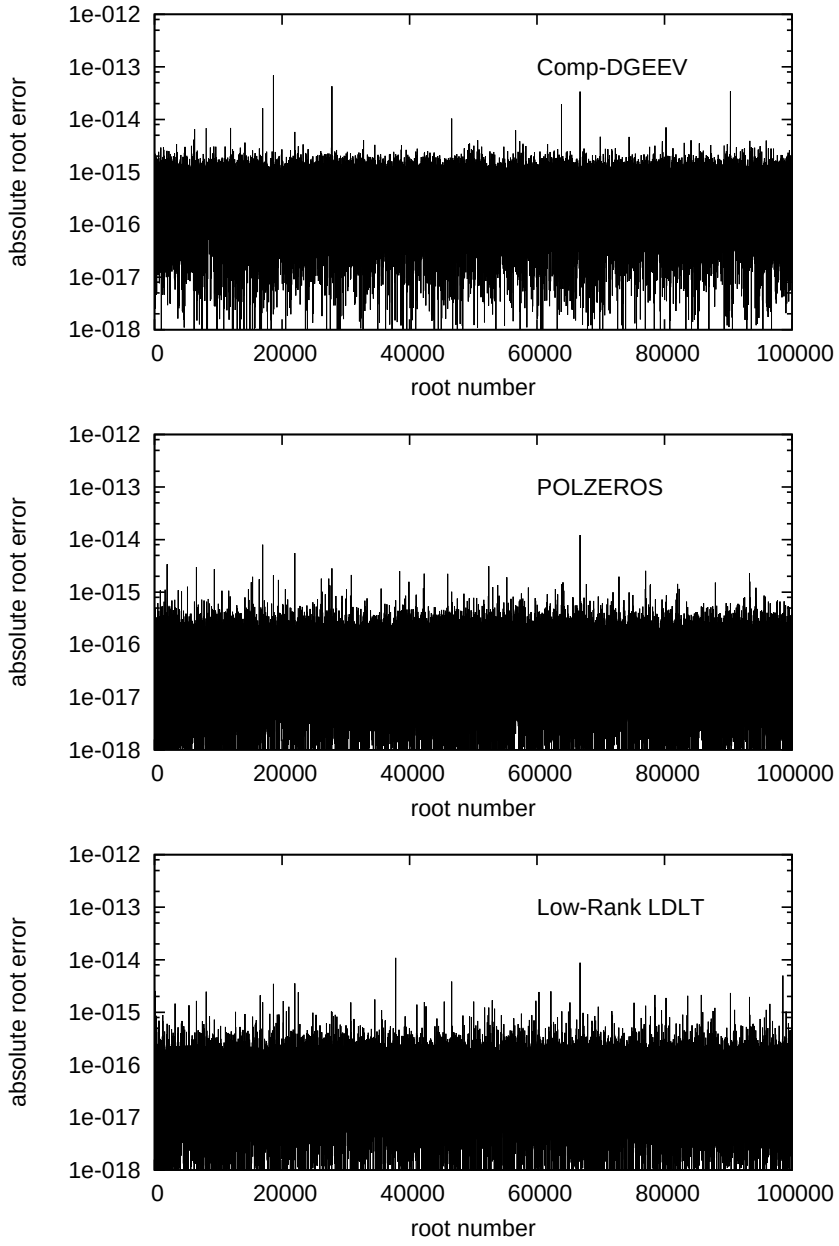

FIG. 6.1. *Absolute root error sequences for the random coefficient scenario in* 25000 *statistically independent trial runs. Top: Companion-eigenvalue method using Lapack dgeev.f. Center: POLZEROS. Bottom: Low-rank* LDL^T *quartic solver.*

We are interested in monitoring the estimation errors of the *minor root* $z_1(k)$. Hence only the sequence of absolute root errors of $z_1(k)$ is recorded and evaluated here. This test comprises 10^5 trial runs because only the minor root error statistics is monitored. Fig. 6.5 shows the corresponding ROC-curves of the minor root estimation errors. LDLT and POLZEROS come up as the top performers and POLZEROS

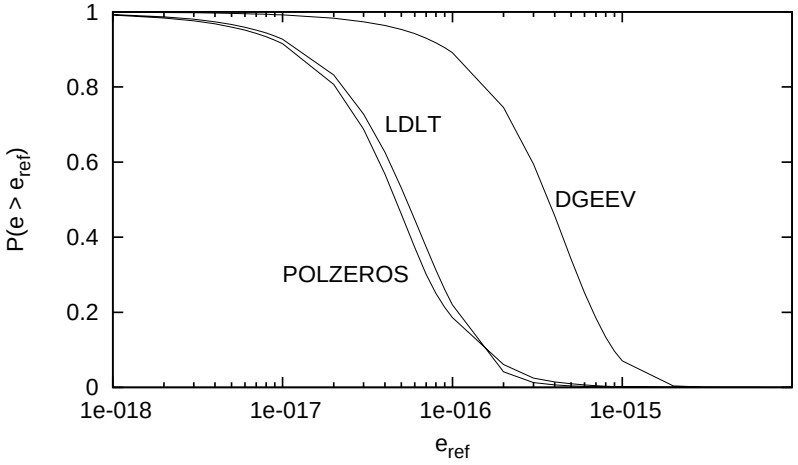

FIG. 6.2. *Probability of exceeding error $P(e > e_{ref})$ or ROC-curves for the 3 error tracks shown in Fig. 6.1.*

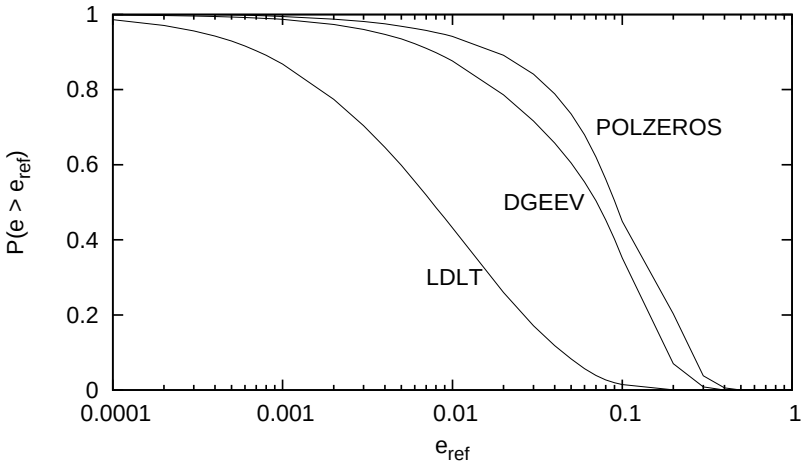

FIG. 6.3. *ROC curves for the clustered real roots scenario with* 25000 *statistically independent trial runs.*

performs slightly better than LDLT. DGEEV produces a poor result, with a loss of approximately 3 decimal digits on average compared to LDLT and POLZEROS. The reason is the known relative inaccuracy of the QR algorithm in estimating tiny roots.

6.4. Runtimes. Another aspect of interest are the runtimes of the algorithms under test. We measured the runtimes in seconds for 10^6 trial runs in each scenario using the `CPU_time` command in Fortran. The results are displayed in Table 6.1. We can see from Table 6.1 that the runtimes of DGEEV and POLZREOS are clearly scenario dependent. In demanding scenarios the runtime of these algorithms can grow by a factor of 3 approximately compared to the runtimes in less demanding scenarios. This is a typical characteristics of iterative algorithms. On the other hand, we observe

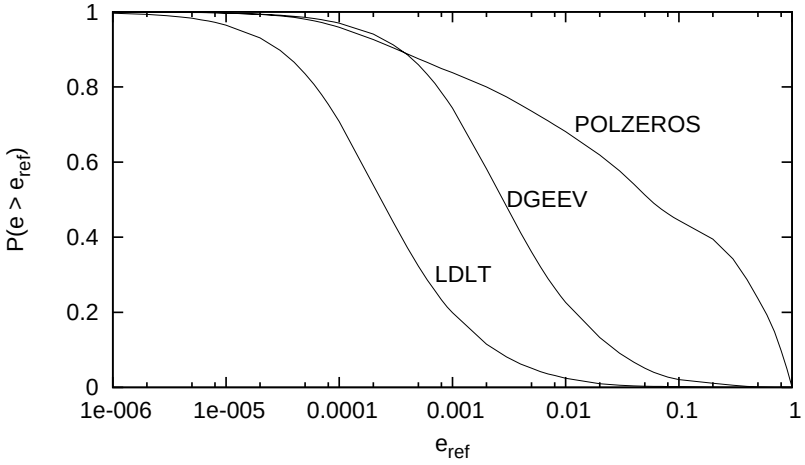

FIG. 6.4. *ROC curves for the clustered complex conjugate roots scenario with* 25000 *statistically independent trial runs.*

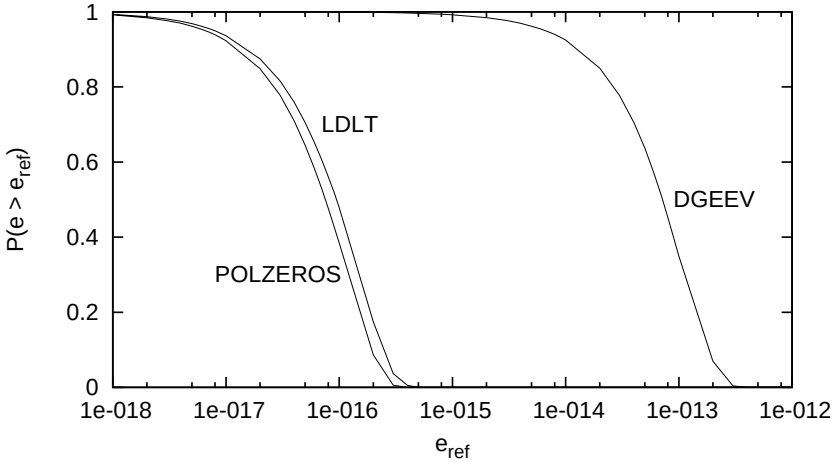

FIG. 6.5. *ROC curves for the large spread scenario with* 10^5 *statistically independent trial runs.*

that in the case of the LDLT algorithm, the runtimes show much less variations and appear clearly not increased by demanding scenarios. Overall, LDLT runs much faster than the general purpose root finders. Variations of the measured runtimes in Table 6.1 are within the statistical fluctuations of the `CPU_time` counter under Fortran.

6.5. Some Extreme Benchmark Tests. We shall conclude this experimental Section by showing the results of some extreme benchmark tests. These are single trial runs.

6.5.1. The Quadruple Root Test. An obligatory test is a run with a quadruple root quartic. We try one with a quadruple root at $z_q = 1000$. The following results are obtained:

Scenario	DGEEV	POLZEROS	LDLT
random coefficients	4.695 sec	3.198 sec	0.6552 sec
clustered real roots	12.32 sec	8.065 sec	0.6240 sec
clustered complex roots	11.63 sec	8.065 sec	0.5928 sec
roots with large spread	3.556 sec	2.542 sec	0.5616 sec

TABLE 6.1

Runtimes for 10^6 trial runs of the algorithms under test in different scenarios on a ThinkPad W520 (Intel Core i7).

```
----- Companion Matrix using Lapack dgeev.f ------
(999.858663364066,0.141346485753255)
(999.858663364066,-0.141346485753255)
(1000.14133663593,0.141326782073270)
(1000.14133663593,-0.141326782073270)

----------------- POLZEROS ---------------------
(999.780644510321,6.335405367989611E-002)
(999.949692471958,-0.180687936306398)
(1000.08129851090,0.303118746700735)
(1000.24987721631,-6.321370229524327E-002)

---------- Low-Rank LDLT Quartic Solver ----------
(1000.00000000000,1.525878906250000E-005)
(1000.00000000000,-1.525878906250000E-005)
(1000.00000000000,0.000000000000000E+000)
(1000.00000000000,0.000000000000000E+000)
```

Both DGEEV and POLZEROS loose about 3/4 of the mantissa digits. Additionally, both methods produce relatively large erronous imaginary parts in the root estimates. LDLT performs uncomparably better with perfect estimates of the real parts of all roots. One root-pair has a spurious imaginary component in the range 10^{-5}.

6.5.2. The Triple Root Test. This test shows the results produced by the algorithms when applied to a quartic with a triple root at $z_\tau = 1000$ and a minor root at $z_0 = 10^{-15}$.

```
----- Companion Matrix using Lapack dgeev.f ------
(0.000000000000000E+000,0.000000000000000E+000)
(999.989897079782,0.000000000000000E+000)
(1000.00505146011,8.749420650332386E-003)
(1000.00505146011,-8.749420650332386E-003)

----------------- POLZEROS ---------------------
(1.000000000000000E-015,9.860761315262648E-032)
(999.994747309788,-7.725111102950707E-003)
(999.993130649025,1.488742047131307E-002)
(1000.01227900630,-1.030412143522037E-003)
```

```
---------- Low-Rank LDLT Quartic Solver ----------
(1000.00000000000,0.000000000000000E+000)
(1.000000000000000E-015,0.000000000000000E+000)
(1000.00000000000,0.000000000000000E+000)
(1000.00000000000,0.000000000000000E+000)
```

We can see that DGEEV crunches the tiny root completely away and replaces it by a plain zero. This shows again the known weakness of the practical QR algorithm in representing tiny roots with a sufficient relative accuracy. POLZEROS estimates the tiny root accurately, but looses more than half of the mantissa digits in estimating the dominant triple root. LDLT produces a perfect result without any error.

6.5.3. The Kahan Benchmark Test. In a recent note [14], W. Kahan proposed the following extreme spread test for quartic root finder algorithms. For a given large number S, with $S > 10^{13}$, compute the roots of a quartic with coefficients

$$A = -(1 + 1/S) \quad , \tag{6.14}$$

$$B = 1/S - S^2 \quad , \tag{6.15}$$

$$C = S^2 + S \quad , \tag{6.16}$$

$$D = -S \quad . \tag{6.17}$$

The ideal roots are $[-S, 1/S, 1, S]$. We ran this test with our 3 algorithms for a parameter S fixed to $S = 10^{15}$. The following results were obtained:

```
----- Companion Matrix using Lapack dgeev.f ------
(9.992007221626409E-016,0.000000000000000E+000)
(0.999999999999997,0.000000000000000E+000)
(-1.000000000000000E+015,0.000000000000000E+000)
(1.000000000000000E+015,0.000000000000000E+000)

------------------ POLZEROS ----------------------
(9.999999999999999E-016,0.000000000000000E+000)
(1.00000000000000,1.110223024625157E-016)
(1.000000000000000E+015,-3.552713678800501E-015)
(-1.000000000000000E+015,-8.940696716308594E-008)

---------- Low-Rank LDLT Quartic Solver ----------
(1.000000000000000E+015,0.000000000000000E+000)
(-1.000000000000000E+015,0.000000000000000E+000)
(1.00000000000000,0.000000000000000E+000)
(1.000000000000000E-015,0.000000000000000E+000)
```

DGEEV produces a robust result, as expected, but the estimated tiny root has only 3 accurate mantissa digits. POLZEROS estimates the real parts of the roots almost perfectly, but produces unacceptably high spurious imaginary parts. LDLT produces a perfect result without any error.

6.5.4. Another Extreme Spread Test. Suppose we wish to estimate a tiny complex conjugate root pair at $z_{1/2} = 1.0 \pm i\,0.1$ in presence of a dominant complex conjugate root pair at $z_{3/4} = 10^{14} \pm i\,10^{7}$. The following results are obtained:

```
----- Companion Matrix using Lapack dgeev.f ------
(0.999999999999791,0.100000000001888)
(0.999999999999791,-0.100000000001888)
(100000000000000.,10128506.0441057)
(100000000000000.,-10128506.0441057)

------------------ POLZEROS ----------------------
(1.00000000000000,9.999999999999944E-002)
(1.00000000000000,-0.100000000000000)
(100000000005990.,10100962.2680463)
(100000000209590.,-9799427.72154323)

----------- Low-Rank LDLT Quartic Solver ---------
(1.00000000000000,0.100000000000000)
(1.00000000000000,-0.100000000000000)
(100000000000000.,10057587.6707783)
(100000000000000.,-10057587.6707783)
```

We can see that DGEEV looses mantissa digits in both the real and the imaginary parts of the tiny root pair. Additionally, there is a heavy loss of mantissa digits in the imaginary part of the dominant root pair. POLZEROS produces a clearly better result in estimating the tiny root pair but looses heavily in estimating the dominant root pair. Moreover, we observe that POLZEROS in the form in which it is available, does not force the estimates to exactly complex conjugate pairs as it should be in the case of real coefficients. LDLT produces a perfect estimate of the tiny root pair and shows as well a loss of digits in the imaginary part of the dominant root pair. However, this loss is not quite as drastic as observed with the two standard methods.

6.5.5. A Case of Clustered Roots. Finally, we look at an experiment using a quartic with graded clustered real roots. These roots assume the locations $z_1 = 30000$, $z_2 = 30001$, $z_3 = 30010$, and $z_4 = 30100$. The following results are obtained:

```
----- Companion Matrix using Lapack dgeev.f ------
(30000.3013835296,1.81298256560417)
(30000.3013835296,-1.81298256560417)
(30010.4011967873,0.000000000000000E+000)
(30099.9960361535,0.000000000000000E+000)

------------------ POLZEROS ----------------------
(29999.2106586734,0.824164617586320)
(30001.1693954703,-0.443021702819903)
(30009.4508102282,0.357452469824679)
(30100.0003705718,9.354120767204877E-007)

---------- Low-Rank LDLT Quartic Solver ----------
(30001.2913551137,0.000000000000000E+000)
(29999.7485371820,0.000000000000000E+000)
(30100.0004673796,0.000000000000000E+000)
(30009.9478702746,0.000000000000000E+000)
```

We can see that DGEEV shows a severe malfunction: The two tightly clustered roots at $z_1 = 30000$ and $z_2 = 30001$ appear merged to a complex conjugate root pair. The other two less tightly clustered roots are estimated with 5 mantissa digits accuracy. POLZEROS shows again the property that it estimates the real parts of the roots quite accurately, at least up to 5 mantissa digits accuracy, but produces again unacceptably large spurious imaginary parts in the root estimates. LDLT produces the overall best result.

6.5.6. A Seemingly Simple Case. Concluding this Section, we show the results obtained for an intuitively uncritical case with two complex conjugate root pairs located at $z_{1/2} = 4 \cdot 10^5 \pm i\,3 \cdot 10^2$ and $z_{3/4} = 3 \cdot 10^4 \pm i\,7 \cdot 10^3$. We obtain:

```
----- Companion Matrix using Lapack dgeev.f ------
(30000.0000000000,7000.00000000027)
(30000.0000000000,-7000.00000000027)
(400000.000000000,300.000000207490)
(400000.000000000,-300.000000207490)

------------------ POLZEROS ----------------------
(30000.0000000000,7000.00000000001)
(30000.0000000000,-7000.00000000000)
(399999.999999569,300.000000273273)
(400000.000000000,-299.999999974002)

---------- Low-Rank LDLT Quartic Solver ----------
(30000.0000000000,7000.00000000001)
(30000.0000000000,-7000.00000000001)
(400000.000000000,300.000000000000)
(400000.000000000,-300.000000000000)
```

Both DGEEV and POLZEROS show an amazing loss of 6 mantissa digits in the estimates of the smaller imaginary parts. LDLT is not puzzled by this case and produces the expected almost perfect estimates.

7. Conclusions. The low-rank LDL^T quartic solver is the ultimate linear algebra answer to the quartic factorization problem. It arises from the sudden insight that the coefficient vectors $[1, a, b]^T$ and $[1, c, d]^T$ of the quadratic factors of a quartic must span a 2-dimensional subspace of a 3-dimensional space in the regular case, or even only a 1-dimensional subspace of that 3-dimensional space in exceptional cases like quadruple roots. Consequently, it is clear that all the information that exists in a quartic must be representable in a matrix of dimension 3×3, not in a matrix of dimension 4×4, as known from Companion matrix methods. These methods are therefore inherently *suboptimal*, because they operate in a space of unnecessarily large dimension. Starting out from these insights, we only had to demonstrate how we could bring the concept to work in a practical algorithm. This is shown in Section 2 but the plain theory would not result in an algorithm of high quality if just we program these equations down in the "raw" fashion in which they appear from the algorithm development. There are too many cases of potential cancellation of significant mantissa digits by subtraction of equally signed numbers. Fortunately, the LDL^T concept is sufficiently compact and free of transcendental operations. This has opened us many

doors for bypassing these critical subtractions of equally signed numbers. The result is a reliable, fast and accurate quartic solver with scenario-independent runtimes. Our Fortran subroutine implementation of the method is released together with the paper.

REFERENCES

[1] N.H. Abel, "Beweis der Unmöglichkeit, algebraische Gleichungen von höheren Graden als dem Vierten allgemein aufzulösen." J. reine angew. Math. 1, 65, 1826. Reprinted in Abel, N. H. (Ed. L. Sylow and S. Lie). Christiania [Oslo], Norway, 1881. Reprinted in New York: Johnson Reprint Corp., pp. 66-87, 1988.

[2] D. Herbison-Evans, "Solving quartics and cubics for graphics", Technical Report TR94-487, Basser Dept. Computer Science, Univ. of Sydney, 2004.

[3] J. D. Foley, A. Van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practice*, Addison-Wesley, 1995.

[4] G.B. Hughes and M. Chraibi, "Calculating ellipse overlap areas", Comput. Visual Sci., Vol. 15, pp. 291-301, Jan. 2014.

[5] M. Abramowitz, and I.A. Stegun (Eds.), "Solutions of quartic equations." §3.8.3 in *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, 9th printing, New York: Dover, pp. 17-18, 1972.

[6] P. Borwein and T. Erdelyi, "Quartic Equations." §1.1.E.1e in *Polynomials and Polynomial Inequalities*, New York: Springer-Verlag, p. 4, 1995.

[7] P. Strobach, "The fast quartic solver", Journal of Computational and Applied Mathematics, Vol. 234, pp. 3007-3024, 2010.

[8] W.H. Press: *Numerical Recipes in FORTRAN 77: The Art of Scientific Computing*, second ed., Cambridge University Press, Cambridge, MA, 1992.

[9] E. Anderson, et al., LAPACK User's Guide, third ed., SIAM Customer Service, Pittsburgh, PA, 1999.

[10] D.A. Bini, "Numerical computation of polynomial zeros by means of Aberth's method", *Numerical Algorithms*, Vol. 13, pp. 179-200, 1996.

[11] D.A. Bini and G. Fiorentino, "MPSolve homepage", http://www.dm.unipi.it/cluster-pages/mpsolve/index.htm.

[12] H. L. VanTrees, *Detection, Estimation, and Modulation Theory: Part 1*, John Wiley & Sons, New York, 2001.

[13] Z. Zeng, "Computing multiple roots of inexact polynomials", Mathematics of Computation, Vol. 74, pp. 869-903, 2005.

[14] W. Kahan, "How (not) to solve a real quartic", Lecture Note Scientific and Engineering Computation Seminar, www.eecs.berkeley.edu/ wkahan/Math128/5Mar14.pdf, Berkeley, CA, March 2014.