Article

Not peer-reviewed version

Computation of Approximate Symmetric Chordal Metric for Complex Numbers

Vasile Sima *

Posted Date: 23 May 2025

doi: 10.20944/preprints202505.1817.v1

Keywords: algorithm; block diagonalization; software; chordal metric

 doi:10.20944/preprints202505.1817.v1

Article

Computation of Approximate Symmetric Chordal Metric for Complex Numbers

Vasile Sima

Technical Sciences Academy of Romania, Bucharest, Romania; vasilesima@ymail.com

Abstract: The basic theoretical properties of the approximate symmetric chordal metric (ASCM) for two real or complex numbers is studied, and reliable, accurate and efficient algorithms are proposed for its computation. ASCM is defined as the minimum between the moduli of the differences of the two numbers and of their reciprocals. It differs from chordal metric by including the modulus of the difference of the numbers. Is is shown that ASCM is not a true mathematical distance, but a useful replacement for a distance in some applications, such as block diagonalization of matrix pencils. A factored representation of the modulus of complex numbers is introduced, which allows to obtain very accurate results for the entire range of the floating point number system of a computer. Two different ways to evaluate the distance between the reciprocals of the given numbers are described. The algorithms can be easily implemented on various architectures and compilers. Extensive numerical tests have been performed to assess the performance of the associated implementation. The results have been compared to those obtained in MATLAB, but with appropriate modifications for numbers very close to the bounds of the range of representable values, where usual formulas give wrong results.

Keywords: algorithm; block diagonalization; software; chordal metric

1. Introduction

The *approximate symmetric chordal metric* for two real or complex numbers is defined by

$$d(a_1, a_2) := \min(|a_1 - a_2|, |1/a_1 - 1/a_2|) =: \min(d_1, d_2). \tag{1}$$

It differs from the Euclidean distance by also considering the absolute difference of the reciprocals of the given numbers. As shown in Remark 1, Section 2, d in (1) is not a true mathematical distance, but a useful replacement for a distance in some applications. In particular, it can be used even if one number, say a_2, is infinite; as shown in Theorem 1 (vi), the result is finite if $a_1 \neq 0$.

There are quite many references on chordal metric, but no references to the approximate symmetric chordal metric could be found. In one of the early papers on chordal metric, Klamkin and Meir [1] have shown that a normed linear space is Ptolemaic if and only if it is symmetric, and that the chordal metric, defined using that space norm, $d(x, y) = \|x - y\| / \|x\| / \|y\|$, for $x, y \neq 0$, satisfies the triangle inequality. A main approach to define the chordal metric is via the stereographic projection, e.g., [2–4]. Let $S^2 \subset \mathbb{R}^3$ be the Riemann sphere with diameter 1 and center in $(0, 0, 1/2) \in \mathbb{R}^3$, defined by $u^2 + v^2 + (w - 1/2)^2 = (1/2)^2$. The uv-plane tangent to S^2 in the origin can be seen as the complex z-plane $\mathbb{C}$. Let $\imath$ be the purely imaginary unit. *Stereographic projection* is the map $\mathbb{C} \to S^2$ defined by the intersection of the chord $\overline{zN}$ with S^2, where $z := (u, v, 0)$ is identified with $z = u + v\imath =: \Re(z) + \Im(z)\imath$ in the uv-plane, and $N := (0, 0, 1)$ is the north pole of S^2. Let $\mathcal{L}(t) := (tu, tv, 1 - t)$ be a parametrization of the line from N to z, with $t \in \mathbb{R}$; clearly, $\mathcal{L}(0) = N$ and $\mathcal{L}(1) = (u, v, 0) = z$; hence, the intersection $S^2 \cap \mathcal{L}(t)$ is for $t_z \in [0, 1)$, and t_z satisfies the equation $(u^2 + v^2 + 1)\, t_z^2 - t_z = 0$. The solutions are $t_z = 0$, corresponding to N, and $t_z = 1/(u^2 + v^2 + 1)$, giving $\mathcal{L}(t_z) = (u, v, u^2 + v^2)/(u^2 + v^2 + 1)$. The stereographic projection of $\mathbb{C}$ to S^2 is defined by

$$\chi : \mathbb{C} \to S^2 \setminus N : z \mapsto \frac{(\Re(z), \Im(z), |z|^2)}{1 + |z|^2}.$$

 doi:10.20944/preprints202505.1817.v1

If $d(z,w) = |z-w|$ is the Euclidean metric in $\mathbb{C}$ and $d(\chi(z),\chi(w))$ that on S^2 inherited from $\mathbb{R}^3$, then it can be shown that,

$$d(\chi(z),\chi(w)) = \frac{|z-w|}{\sqrt{1+|z|^2}\sqrt{1+|w|^2}}, \tag{2}$$

which is the *chordal metric* of S^2. Clearly, $d(\chi(z),\chi(w)) = 0 \Leftrightarrow d(z,w) = 0$. If $z \to \infty$, then $\chi(z) \to (0,0,1) = N$. This way, χ becomes a map from the extended complex plane $\overline{\mathbb{C}} = \mathbb{C} \cup \{\infty\}$ to S^2. Note that $\chi(z,w) = \chi(1/z, 1/w)$ if z, $w \in \mathbb{C}$. The triangle inequality is satified by $d(\chi(z),\chi(w))$, being implied by that inequality for Euclidean distance in $\mathbb{R}^3$; therefore, chordal metric is indeed a distance. If $w \to \infty$, then the formula becomes $d(z,\infty) = 1/\sqrt{1+|z|^2}$. If a Riemann sphere with unit radius and center in the origin is used, then the formulas for the chordal metric should be multiplied by 2 [3,5]. It is also mentioned in [3] that $\chi(z,w) = 0$ if $z \to \infty$ and $w \to \infty$.

Several authors present theretical results and various applications of the chordal metric. Rainio and Vuorinen [4] use (2) to identify some cases where the distortion caused by symmetrization of a quadruple of points or objects, mapped by a Möbius transformation, can be measured in terms of the Lipschitz constant of this transformation in the Euclidean or the chordal metric. Álvarez-Tuñón et al [6] investigate various pose representations and metric functions in visual odometry networks, and show that the chordal distance ensures better generalization and faster convergence of the network parameters than distances measured by Euler angles or quaternions; this is useful to estimate the change of position, e.g., of robots, relative to a starting location, based on data from motion sensors. However, this paper uses the chordal distance between two rotations, R_A and R_B, defined as $\|R_A - R_B\|_F = \|R_A R_B^T - I\|_F$, where I is the identity matrix of order 3 and F denotes the Frobenius norm. Xu et al [7] provide explicit expression and sharper bounds of the chordal metric between generalized singular values of Grassmann matrix pairs. A constrained optimization problem of the form $\max_{U \in \mathbb{U}_n} f(U)$, where $\mathbb{U}_n$ is the set of $n \times n$ unitary matrices, is studied and its optimal solution is given. The results are applied to comparative analysis of gene mRNA expression data for mice macrophage under different conditions. Sun [8] considers the generalized singular values (GSVD) when $\text{rank}([A;B]) < n$, where A and B are complex $m \times n$ and $p \times n$, respectively. Then, any pair (a,b) satisfying $a,b \geq 0$ and $(a,b) \neq (0,0)$ is a singular value, a/b. For such a case, it is difficult to investigate the perturbation of singular values. It is shown that when $[\ A;B\ ]$ is strongly perturbed, the singular values are insensitive to perturbations in the elements of A and B, if the chordal metric is used to describe such perturbations. Sasane [2] defines an abstract chordal metric on linear control systems described by their transfer functions and shows that strong stabilizability is a robust property in this metric. In [9], a refinement of this metric for standard classes of stable transfer functions is introduced.

A common application of the approximate symmetric chordal metric is related to ordering the eigenvalues of a matrix or matrix pencil, according to their pairwise distances measured using (1). For generalized eigenvalue problems, i.e., for matrix pencils, some eigenvalues may be infinite. Ordering (generalized) eigenvalues is very important in many applications of numerical linear algebra, control theory and other domains. For instance, it is essential for computing invariant or deflation subspaces of matrices or matrix pencils, respectively. A more general application of the approximate symmetric chordal metric is for computing the block-diagonal form for matrices or matrix pencils [10], especially for large order problems. This form is useful for finding reciprocal condition numbers for eigenvalues and related eigenspaces, as well as for fast computation of matrix functions or of the responses of linear time invariant standard or descriptor systems. Assuming that the matrix (pencil) is already reduced to a (generalized) Schur form [11], an essential part of the computations is the solution of a (generalized) Sylvester equation. This part involves non-unitary transformations. To ensure an acceptable numerical precision, the solution elements should be bounded. Specifically, if any element of the solution exceeds a given threshold $\tau > 1$, then the process is stopped, and an enlarged Sylvester equation is tried to be solved. If many failed attempts are necessary, then the computational effort might be unacceptably high, especially for large order problems. A clever strategy for ordering the eigenvalues of the matrix (pencil) might reduce the number of failed trials, since the magnitude of the solution is influenced

 doi:10.20944/preprints202505.1817.v1

by the sensitivity of the spectrum associated to the current block that should be separated. The sensitivity of eigenvalue problems and related topics are investigated in [11–16]. Efficient and reliable implementations of the theoretical results are available in the LAPACK package [17] and in interactive environments like MATLAB [18] or Mathematica [19]. The eigenvalues can be ordered based on their pairwise "distances" measured by (1). These eigenvalue applications do not need the triangle inequality be satisfied. But it is important that close eigenvalues be included in the same cluster. The use of the min function in (1) will choose an eigenvalue a_2 to be added to a cluster containing a value a_1 if either $|a_1 - a_2|$ or $|1/a_1 - 1/a_2|$ is smaller than a given threshold. An approach based on the chordal metric cannot do that. Clearly, $d_2 = |1/a_1 - 1/a_2| = |a_1 - a_2|/|a_1|/|a_2|$, if $a_1 \neq 0$ and $a_2 \neq 0$. Therefore, in such a case, d_2 coincides to the chordal metric in [1], and d_2 is often quite close to the usual definition of the chordal metric (2). The distance $d_1 = |a_1 - a_2|$ is not directly taken into account, although it could be smaller than d_2.

This section is ended by some issues, needed in the sequel, concerning the floating point system of computers. For technical convenience, $\overline{\mathbb{R}}$ will denote the real numbers field $\mathbb{R}$ extended by the ∞ and $-\infty$ symbols, and similarly for $\overline{\mathbb{C}}$. This is useful, since practical calculations with computer software like MATLAB encounter such situations.

A complex "number" $a = \alpha + \beta\imath$, with $\alpha, \beta \in \overline{\mathbb{R}}$, is *infinite* if $\alpha = \pm\infty$ and/or $\beta = \pm\infty$. Otherwise, a is finite. Consider two finite complex numbers, $a_1 = \alpha_1 + \beta_1\imath$, $a_2 = \alpha_2 + \beta_2\imath$. It is assumed that α_i and β_i, $i = 1, 2$, are in the range of the *floating point* system of a computer, hence a_1 and a_2 are (approximately) *representable* by their real and imaginary parts. Therefore, $|\alpha_i|, |\beta_i| \leq K$, $i = 1, 2$, where K is the maximum representable number, defined by

$$K = b^{e_M}(1 - \epsilon), \tag{3}$$

where b is the base of the numeral system, e_M is the maximum exponent, and ϵ is the smallest distance between two representable numbers (also called the *relative machine precision*). In a commonly used binary double precision floating point system, $b = 2$, $e_M = 1024$, $\epsilon = 2^{1-t}$ (or $\epsilon = 2^{-t}$ for machines with *proper rounding* in the addition operation), and $t = 53$, where t is the number of base digits in the *mantissa*, i.e., the fractional part of the representation in base b. Note that the result of evaluating K as in (3) might not be representable on such a computing machine, since it could exceed K; but evaluating $K = \left(b^{e_M - 1}(1 - \epsilon)\right) \times 2$ will not overflow. The number $\varepsilon = b\epsilon$ is called *precision*. On the machine considered above, with proper rounding, $(1 + \epsilon) - 1 = 0$, but $(1 + \varepsilon) - 1 = \varepsilon$.

Clearly, not all real or complex numbers can be represented on a digital computer, since the floating point system deals with rational approximations of the numbers in the range $\left[-K, K\right]$. If an arithmetic operation would produce a result outside this range, that fact is called *overflow*. Compilers and interactive environments may represent such a result as $\pm$`Infinity` or $\pm$`Inf`, and if this happens in a sequence of computations, the final result might be completely wrong. Therefore, overflows should be avoided. On the other hand, very small (in magnitude) numbers may also be not representable, and if this is the case for the result of a mathematical operation, it may be set to zero. This is the counterpart of an overflow, and it is called *underflow*, but it is not as severe as overflows. The underflow threshold is defined by $k = b^{e_m - 1}$, where e_m is the minimum exponent before underflow. For the machine considered above, $e_m = -1021$. Many computer systems allow for *gradual* underflow, i.e., they may also use numbers less than k, in the range $\left[\varepsilon k, k\right)$. For this reason, the availability of gradual underflow will be assumed in this paper.

Algorithms for discovering the properties of the floating point arithmetic, like in [20,21], are implemented, e.g., in the `SLAMCH` and `DLAMCH` routines of the LAPACK package [17], for single and double precision computations, respectively. A special function is used to ensure that all relevant values are stored, and not held in registers, or not affected by compiler optimizations.

In the sequel, there will be no distinction between a number and its floating point representation. Note that the assumption that $|\alpha_i|, |\beta_i| \leq K$, $i = 1, 2$, does not imply that $|a_1| \leq K$ and $|a_2| \leq K$.

 doi:10.20944/preprints202505.1817.v1

If $|\alpha_1| = K$ and $|\beta_1| = K$, then $|a_1| = K\sqrt{2}$, which is not representable. The specific case of our application allows to work with a factored form of $|a_1|$ and $|a_2|$.

The rest of the paper is organized as follows. Section 2 investigates the properties of the approximate symmetric chordal metric, while Section 3 presents reliable, accurate, and efficient algorithms for its computation. Section 4 discusses the numerical results obtained using a careful implementation of these algorithms. Section 5 summarizes the conclusions.

2. Basic Properties of the Approximate Symmetric Chordal Metric

In some applications using (approximate symmetric) chordal metric, a_1 and/or a_2 or the terms in (1) may not be numbers, but $0/0$ or $\infty - \infty$. For instance, some eigenvalues of singular matrix pencils are $0/0$. Moreover, even for nonsingular matrix pencils, some eigenvalues are infinite, but in some routines, e.g., as those in LAPACK [17], the eigenvalues are in fact ratios of pairs of numbers, $\mu \neq 0$ and ν, with ν possibly 0; if $a_1 = a_2 = \infty$ in (1), for $a_1, a_2 \in \overline{\mathbb{R}}$, then the first term of the min function in (1) is $\infty - \infty$. In such cases, it is necessary to use special computer rules for mathematical operations, as shown below.

A Not-a-Number entity is represented by `NaN` in compilers and interactive environments like MATLAB.

Definition 1. *(Basic operations with `NaN`s and numbers)*

- $\texttt{NaN} \circ a = \texttt{NaN}$, $\texttt{NaN}^a = \texttt{NaN}$, $a^{\texttt{NaN}} = \texttt{NaN}$, *where* $\circ \in \{+, -, \times, /\}$, $\times$ *and* $/$ *are the multiplication and division binary operations, and* $a \in \overline{\mathbb{R}}$.
- $\texttt{NaN} \circ a = \texttt{NaN} \circ \beta\imath$, $\texttt{NaN}^a = \texttt{NaN} + \texttt{NaN}\imath$, $a^{\texttt{NaN}} = \texttt{NaN} + \texttt{NaN}\imath$, *for* $a =: \alpha + \beta\imath \in \overline{\mathbb{C}}$.
- $|\texttt{NaN}| = \texttt{NaN}$, $\min(\texttt{NaN}, a) = a$, $\max(\texttt{NaN}, a) = a$, *for* $a \in \overline{\mathbb{C}}$.

For convenience, conventions like $1/0 = \infty$ and $-1/0 = -\infty$ are adopted here; these conventions make sense thinking in terms of limits, for intance, $\lim_{x \to 0} 1/x = \pm\infty$, for x taking positive of negative values only, respectively.

The next theorem shows the basic properties of the approximate symmetric chordal metric. The results are true even if α_i and/or β_i, $i = 1, 2$, are infinite.

Theorem 1 (Basic properties of d in (1))**.** *Let a, a_1 and a_2 be finite or infinite complex numbers. Then, with (1)*

(i) $d(a_1, a_2) = d(a_2, a_1)$;

(ii) $d(a, 0) = |a|$, $\forall a \in \overline{\mathbb{C}}$;

(iii) $d(a, a) = 0$, $\forall a \in \overline{\mathbb{C}}$;

(iv) $d(a_1, a_2) = 0$ *if* a_1 *and* a_2 *are infinite;*

(v) $d(a_1, a_2) = 0 \Leftrightarrow a_1 = a_2, \forall a_1, a_2 \in \overline{\mathbb{C}}$;

(vi) $d(a_1, a_2) = 1/|a_1|$, *if* a_2 *is infinite,* $\forall a_1 \in \overline{\mathbb{C}}$;

(vii) $d(a_1, a_2) > 0$ *if* $a_1 \neq a_2$ *and* a_1 *or* a_2*, or both, are finite.*

Proof. The symmetry property (i) follows immediately from (1). Property (ii) holds since the second argument of the min function in (1) is infinite, hence, $d_2 = \infty$, and $d_1 = |a|$; clearly, (ii) is also true for $a = \pm\infty$, $a = \pm\infty + \beta\imath$, $a = \alpha \pm \infty\imath$, or $a = \pm\infty \pm \infty\imath$, with α and β finite, since then $d_1 = \infty$ too. (For convenience, when $a = \pm\infty \pm \infty\imath$ appears, the other two cases, with either α or β finite, will not be mentioned, but the related property is also true for these cases.) From (1), property (iii) holds for finite a; if $a = \pm\infty$ or $a = \pm\infty \pm \infty\imath$, then $d_1 = \texttt{NaN}$, $d_2 = 0$, and $d = 0$, according to the third item in Definition 1. If $a_1 = \pm\infty$ or $a_1 = \pm\infty \pm \infty\imath$ and $a_2 = a_1$, (iv) has been proved in (iii), while if $a_2 = -a_1$, or $a_2 = \pm\bar{a}_1$, where $\bar{a}$ is the complex conjugate of a, then (iv) follows, since $d_1 = \infty$ and $d_2 = 0$. Clearly,

 doi:10.20944/preprints202505.1817.v1

from (iii) and (iv) it follows that $a_1 = a_2$ implies $d(a_1, a_2) = 0$. Conversely, if $d(a_1, a_2) = 0$, then either $|a_1 - a_2| = 0$, if a_1 and a_2 are finite, or $|1/a_1 - 1/a_2| = 0$, if a_1 and a_2 are infinite, so $a_1 = a_2$. These two facts prove the *identity* property (v). If $a_1 \neq 0$ is finite, (vi) follows since $d_1 = \infty$ and $d_2 = 1/|a_1|$; (vi) is also true if $a_1 = \pm\infty$ or $a_1 = \pm\infty \pm \infty\imath$ since, from (iv), $d(a_1, a_2) = 0 = 1/|a_1|$. If $a_1 = 0$, then (vi) holds since $d_1 = \infty$ and $d_2 = \infty$, so $d = \infty = 1/|a_1|$. If a_1 and a_2 are finite and distinct, both terms of min are strictly positive, which prove (vii); if a_1 is finite, but a_2 is infinite, then d_1 is infinite, hence, by (vi), $d = d_2 = 1/|a_1| > 0$. The symmetry property makes the result true also for a_1 infinite and a_2 finite. □

Remark 1. *Theorem 1 proves that three properties of a distance or metric, namely, symmetry, identity and non-negativity, are satisfied by $d(a_1, a_2)$. But it is easy to verify that the fourth needed property, the triangle inequality, does not hold for (1), therefore the inequality $d(a, c) \leq d(a, b) + d(b, c)$, is not true in general, for $a, b, c \in \mathbb{C}$. Consequently, $d(a_1, a_2)$ is not a* distance *or* metric *function. Moreover, the approximate symmetric chordal metric does not satisfy a scaling property, that is, $d(sa_1, sa_2)$ differs, in general, from $|s|d(a_1, a_2)$, with $a_1, a_2, s \in \mathbb{C}$. For instance, for $a_1 = 1$, $a_2 = 2$, and $s = 2$, $d(a_1, a_2) = 0.5$ and $d(sa_1, sa_2) = 0.25$, but for $s = 4$, $d(sa_1, sa_2) = 0.125$. The ratios $d(sa_1, sa_2)/d(a_1, a_2)$ in the two cases are 0.5 and 0.25, respectively, not 2 and 4. Such scaling property holds for $|a_1 - a_2|$ and for $|1/a_1 - 1/a_2|$, but not necessarily for their minimum.*

In order to be able to accurately work on the entire range of representable complex numbers, a special representation of the absolute value is used, as defined below.

Definition 2. *(Factored representation of the absolute value of a complex number a)*
Let $a = \alpha + \beta\imath \in \mathbb{C}$ with $|\alpha| \leq K$ and $|\beta| \leq K$. Let $M := \max(|\alpha|, |\beta|)$, $m = \min(|\alpha|, |\beta|)$, and $\delta = \sqrt{1 + (m/M)^2}$. Then $|a| = M\delta$ is called the factored representation *of $|a|$.*

From Definition 2 it follows that $\delta \in \left[1, \sqrt{2}\right]$, since $\delta = 1$ if $m = 0$, $\delta = \sqrt{2}$ if $m = M$, and values $m/M \in \left(0, 1\right)$ will imply $\delta \in \left(1, \sqrt{2}\right)$. Factored representation is useful when M is vey close to or coincides to K. In such a case, the product $M\delta$ could exceed K, and then $|\alpha|$ cannot be computed. Still, its factors can be used in all computations needed for obtaining the approximate symmetric chordal metric.

3. Algorithms for Computing the Approximate Symmetric Chordal Metric

A conceptual algorithm is first presented, and then it will be detailed. The symbols $\wedge$, $\vee$ and $\neg$, used in the algoriths and their description below, are the binary logical operators `AND`, `OR` and `NOT`, respectively. Moreover, $\overline{\mathbb{C}}_r$ and $\overline{\mathbb{R}}_r$ denote the sets of complex and real representable numbers, respectively. However, the algorithms below will also correctly work for a_1 and/or a_2 set to $\pm$`Infinity` or $\pm$`Infinity` $\pm$`Infinity`$\imath$, if the IEEE arithmetic is available on the computer. Although the computational problem for approximate symmetric chordal metric looks mathematically very simple, a reliable, accurate, and efficient implementation needs a thorough analysis. These desirable properties are supported by careful consideration of all possible special cases and lower level algorithms. Efficiency can be measured by an estimate of the number of *floating point operations*, or *flops*, which have been used especially for numerical linear algebra algorithms [11,22]. One definition says that a flop consists in an addition, a multiplication and few memory addresses. Another definition considers each addition, subtraction, multiplication and division of two floating point numbers as a flop. This is suitable in the paper context.

3.1. Conceptual Algorithm

In the first part of the Algorithm `chordal`, the value of $d = d(a_1, a_2)$ is computed for special cases. If the maximum absolute value, M, of the real and imaginary parts of a_1 and a_2, is zero, or if $a_1 = a_2$, then $d = 0$, according to (iii) in Theorem 1. Otherwise, if $M_1 = 0$, which implies that $a_1 = 0$, then $d = |a_2|$, using (i) and (ii). Similarly, if $M_2 = 0$, then $d = |a_1|$. The absolute values $|a_i|$, for $i = 1$ and $i = 2$, are computed by a lower level algorithm, called `absa`, introduced in the next subsection.

 doi:10.20944/preprints202505.1817.v1

Algorithm `absa` needs on input the auxiliary parameters M and $u = u_1$ or $u = u_2$, set in Algorithm `chordal`, using $\kappa := K/\sqrt{2}$.

Another special case is when one of the numbers $|a_1|$ or $|a_2|$ is negligible compared to the other. Then, as shown below, $|a_1 - a_2|$ can be found using $|a_1|$, $|a_2|$, or both. Detecting this situation involves the logical variable $f := m/M \leq \varepsilon/4$, with $m := \min(M_1, M_2)$. See Remark 2 below. If f is `true`, then it is easier to obtain the terms d_1 and d_2 of d in (1). Specifically, if $M = M_i$ and $j \neq i$, $i, j \in \{1, 2\}$, it follows that: if $m < 0.1/M$, then $d = |a_i|$; else if $m > 1.1/M$, then $d = 1/|a_j|$; else $d = \min\left(|a_i|, 1/|a_j|\right)$. The tests involving m and M are motivated by the following reason: $M = M_i$ implies that $d_1 = |a_i|$ is of the order of M, and $d_2 = 1/|a_j|$ is of the order of $1/m$; if $m \approx 1/M$, d_1 and d_2 have comparable values, and d should be found as $\min\left(d_1, d_2\right)$; if m is smaller enough than $1/M$, then $d_1 < d_2$ and so $d = d_1$. Similarly, if m is larger enough than $1/M$, then $d_1 > d_2$, and so $d = d_2$. The bounds 0.1 and 1.1 have been found after few trials, so that all tests returned the correct results. Note that, in the discussion above and in Algorithm `chordal`, the product mM is not used, since it may overflow if $|a_1|$ and $|a_2|$ are close to K.

Remark 2. *Using Definition 2, the absolute value of a_i, $i = 1, 2$, can be expressed as $|a_i| = M_i\delta_i$, where $\delta_i := \sqrt{1 + \left(m_i/M_i\right)^2} \in \left[1, \sqrt{2}\right]$. Indeed, $\delta_i = 1$, if $m_i = 0$ and $\delta_i = \sqrt{2}$, if $m_i = M_i$. Clearly, if $m_i \in \left(0, M_i\right)$, then $\delta_i \in \left(1, \sqrt{2}\right)$. This* factored form *of $|a_i|$ is valid even if $M_i = m_i = K$, when the factors cannot be multiplied without producing an overflow. If $M_1/M_2 \leq \varepsilon/4$, it follows that*

$$|a_1|/|a_2| \leq (M_1/M_2)(\delta_1/\delta_2) \leq (\varepsilon/4)\sqrt{2} < \epsilon, \tag{4}$$

since $\delta_1 \leq \sqrt{2}$ and $\delta_2 \geq 1$. Therefore, if M_1 is negligible compared to M_2, the same is true for $|a_1|$ and $|a_2|$. The last inequality in (4) assumes that the base of the floating point system is $b = 2$. For the previous paragraph, $M_1 = m$ and $M_2 = M$.

If no special case appears in Algorithm `chordal`, it follows that a_1 and a_2 are nonzero, distinct and relatively "close" to each other, in the sense that $m/M > \varepsilon/4$. This is the general case, for which it is necessary to compute $d_1 = |a_1 - a_2|$ and $d_2 = |1/a_1 - 1/a_2|$. Finally, $d = \min(d_1, d_2)$. These more involved computational steps of the conceptual algorithm are detailed in the next subsections.

An account of the needed computations can be given here for the part involving special cases, but additional details, including those for general case, will be available in the remaining subsections of this section. At the beginning of Algorithm `chordal`, the computation of the absolute values for four real scalars, as well as three *standard* max operations (i.e., with two arguments) are needed, for getting M_i, $i = 1, 2$, and M. If $M = 0$ or $a_1 = a_2$, which requires three comparisons of real scalars, the result is $d = 0$. Otherwise, if $M_i = 0$, then $d = |a_j|$, with $j \neq i$ and $i = 1$ or $i = 2$; each of these cases needs an evaluation of one norm (either of a_1 or of a_2); in addition, a division and two tests are needed to set $u_i = M_i \leq \kappa$, $\kappa := K/\sqrt{2}$. If $M_i \neq 0$ for $i \in \{1, 2\}$, the special case when $|a_1| \gg |a_2|$ or $|a_1| \ll |a_2|$ is tried. The initialization of this pseudocode segment involves a standard min operation, two divisions and a comparison, setting the value of the logical variable f. If $f =$ `true` and $M = M_i$, then there are three cases: if $m < 0.1/M$, then $d = |a_i|$; else, if $m > 1.1/M$, then $d = 1/|a_j|$, for $j \neq i$; else, $d = \min\left(|a_2|, 1/|a_1|\right)$. Hence, in addition to the inialization part, the computations involve three or four tests, one or three divisions, one or two evaluations of the absolute value, and—for the third case—one standard min operation.

 doi:10.20944/preprints202505.1817.v1

Algorithm 1 `chordal` computes ASCM for two complex numbers

Require: $a_1, a_2 \in \overline{\mathbb{C}}_r$, $a_i = \alpha_i + \beta_i \imath$, K, ε.
Ensure: $d := d(a_1, a_2) = \min(|a_1 - a_2|, |1/a_1 - 1/a_2|)$
  $M_i = \max(|\alpha_i|, |\beta_i|), i = 1, 2,\ M = \max(M_1, M_2)$
  if $M = 0$ or $a_1 = a_2$ **then**
    $d = 0$
  else
    $\kappa = K/\sqrt{2},\ u_i = M_i \le \kappa, i = 1, 2$
    if $M_1 = 0$ **then**
      $d = |a_2|$
    else if $M_2 = 0$ **then**
      $d = |a_1|$
    else
      $m = \min(M_1, M_2),\ f = m/M \le \varepsilon/4$
      if f **then**
        if $M = M_1$ **then**
          if $m < 0.1/M$ **then**
            $d = |a_1|$
          else if $m > 1.1/M$ **then**
            $d = 1/|a_2|$
          else
            $d = \min\left(|a_1|, 1/|a_2|\right)$
          end if
        else
          if $m < 0.1/M$ **then**
            $d = |a_2|$
          else if $m > 1.1/M$ **then**
            $d = 1/|a_1|$
          else
            $d = \min\left(|a_2|, 1/|a_1|\right)$
          end if
        end if
      else
        $d_1 = |a_1 - a_2|,\ d_2 = |1/a_1 - 1/a_2|,\ d = \min\left(d_1, d_2\right)$
      end if
    end if
  end if

3.2. Computation of $|a|$, $a \in \overline{\mathbb{C}}_r$

If the magnitude of the real and imaginary parts of $a = \alpha + \beta\imath \in \overline{\mathbb{C}}_r$ are close to the maximum representable number, K, then $|a|$ may overflow. For the computation of the approximate symmetric chordal metric d, it is possible to use the factored form of $|a|$, using M and δ given in Definition 2, namely, $M = \max(|\alpha|, |\beta|)$, and $\delta = \sqrt{1 + (m/M)^2} \in \left[0, \sqrt{2}\right]$, with $m = \min(|\alpha|, |\beta|)$.

For example, for $a = K + K\imath$, the direct computation of `abs(a)` in MATLAB gives `Inf`, the machine infinity, while $|a|$ can very accurately be represented by the product factors $M = K$ and $\delta = \sqrt{2}$. Actually, `abs(a)`, i.e., $\sqrt{\alpha^2 + \beta^2}$, is also evaluated internally in a factored form in MATLAB or in other available routines, like `DLAPY2` from LAPACK [17], but the result is returned as a product, $M\delta$. However, in most cases, $|a|$ is far from K, and always using the factored form would involve some unnecessary computations. For efficiency, a bound is used to detect a case when the factored form might be needed. Specifically, if $M \le \kappa$, with $\kappa := K/\sqrt{2}$ defined in Algorithm `chordal`, then one can safely use `abs(a)` in MATLAB, or `DLAPY2` from LAPACK to obtain $|a|$. Otherwise, it is safer to employ the factored form. Since κ is a conservative bound, it is possible that $|a| = M\delta$ be representable. The needed test is included in the Algorithm `absa`, listed below, and explained in Remark 3; the logical variable u, externally initialized to `true` if $M \le \kappa$, is updated internally as $u = M \le K/\delta$.

 doi:10.20944/preprints202505.1817.v1

Algorithm 2 `absa` computes the modulus (or its factored representation) of a complex number

Require: $a = \alpha + \beta\imath \in \overline{\mathbb{C}}_r$, M, K, u
Ensure: $|a|$, if u is true on exit, or M and δ, otherwise
  if u **then**
    $|a| = \sqrt{\alpha^2 + \beta^2}$
  else
    $m = \min(|\alpha|, |\beta|)$, $\delta = \sqrt{1 + (m/M)^2}$, $u = M \leq K/\delta$
    if u **then**
      $|a| = M\delta$
    end if
  end if

If $u =$ `true` on entry, Algorithm `absa` needs in principle to square two real numbers and make an addition and a square root. This computation can be performed using `abs`(a) in MATLAB, or by a call to the LAPACK routine `DLAPY2`. This is preferable since there are professional implementations of this routine on many computing platforms, that could return a more accurate result than a direct evaluation of the the formula $\sqrt{\alpha^2 + \beta^2}$. If $u =$ `false` on entry to Algorithm `absa`, then the computations involve a standard min operation, the absolute values of two real numbers, two divisions, an addition, a square of a number, a square root and a test $u = M \leq K/\delta$. If $u =$ `true` after this test, then a multiplication is done, in order to simplify the subsequent calculations. Actually, `DLAPY2` uses the factors M and δ as in the **else** part of the algorithm, but then always multiplies them; therefore, the actual operations are the same as above, but, in extreme cases, the product $M\delta$ may overflow.

The algorithm `absa` can be used to compute $|a_1|$, $|a_2|$, $d_1 = |a_1 - a_2|$, and $d_2 = |1/a_1 - 1/a_2|$ in Algorithm `chordal`. For d_2 it is necessary to first evaluate $1/a_1$ and $1/a_2$. This topic will be dealt with in Section 3.5.

Remark 3. *The logical variable u_i is initialized to* `true` *in Algorithm* `chordal` *for a_i, if $M_i \leq \kappa$, $i = 1, 2$. If u_i is* `false` *on entry to Algorithm* `absa`*, but if, after computing the corresponding δ_i, it follows that $M_i \leq K/\delta_i$, then u_i is reset to* `true`*, using $u_i = M_i \leq K/\delta_i$, and hence $|a_i| = M_i\delta_i$. Otherwise, $|a_i|$ is represented and used in the factored form. For instance, for finding $|a_1|$ or its factored representation, the following MATLAB-like command can be used,*

$$[u_1, \mu_1, \delta_1] = \texttt{absa}(a_1, M_1, K, u_1)$$

where $\mu_1 = |a_1|$, if $u_1 =$ `true` on exit from `absa`*, and M_1 and δ_1 define the factored representation of $|a_1|$, if $u_1 =$ `false` on exit. Note that the closeness to the underflow threshold, k, is not taken into account in Algorithm* `absa`*, since underflow is less harmful than overflow. If $M\delta$ underflows, then the product might become 0 instead of a very small positive value. Having a machine with gradual underflow is helpful in such a case, because some nonzero values smaller than k are representable.*

3.3. Computation of $d_1 = |a_1 - a_2|$, $a_1, a_2 \in \overline{\mathbb{C}}_r$

The computation of d_1 needs the real and imaginary parts of $a_1 - a_2$, and these parts are computed separately. Using triangle inequality for the Euclidean distance between the points a_1 and a_2 in the complex plane, it follows that $|a_1 - a_2| \leq |a_1| + |a_2|$. Denoting $M_{12} = \max(|\alpha_1 - \alpha_2|, |\beta_1 - \beta_2|)$, the same inequality expressed in the infinity norm, $|a_i|_\infty = M_i$, $i = 1, 2$, implies that $M_{12} \leq M_1 + M_2$. Clearly, even simple operations like $\alpha_1 - \alpha_2$ or $\beta_1 - \beta_2$ may produce overflow or underflow, if any of these scalars is close to K or k, respectively, and the other has appropriate sign and magnitude. These exceptions can be avoided in a professional implementation, but it is worth to estimate their possible occurrence. This is done in an algorith that computes the difference $\sigma := \alpha - \gamma$ between two real numbers, α and γ. If IEEE arithmetic is available, and either $\alpha_1 - \alpha_2$ or $\beta_1 - \beta_2$ are/is $\pm$`Infinity`, then the corresponding d_1 could not be computed, but d_2 will certainly be. Without IEEE arithmetic, the computation is more involved, as shown in Algorithm `subtract` below.

 doi:10.20944/preprints202505.1817.v1

Algorithm 3 `subtract` computes the difference of two complex numbers

Require: $\alpha, \gamma \in \overline{\mathbb{R}}_r$, K, ε.
Ensure: $\sigma = \alpha - \gamma$, ϕ
  $M = \max(|\alpha|, |\gamma|)$, $\phi =$ `true`
  if $M = 0$ **then**
    $\sigma = 0$
  else
    $m = \min(|\alpha|, |\gamma|)$
    if $m/M \leq \varepsilon/4$ **then**
      if $m = |\alpha|$ **then**
        $\sigma = -\gamma$
      else
        $\sigma = \alpha$
      end if
    else
      $s_\alpha = \text{sign}(\alpha)$
      if $s_\alpha = \text{sign}(\gamma)$ **then**
        $\sigma = \alpha - \gamma$
      else
        if $(\ \gamma = \text{sign}(\gamma)K \wedge |\alpha| > K\varepsilon/4\ ) \vee (\ |\alpha| > K + s_\alpha\gamma\ )$ **then**
          $\sigma = s_\alpha K$, $\phi =$ `false`
        else
          $\sigma = \alpha - \gamma$
        end if
      end if
    end if
  end if

The Algorithm `subtract` uses the sign function, defined as $\text{sign}(\alpha) = 1$, if $\alpha > 0$, $\text{sign}(\alpha) = 0$, if $\alpha = 0$, and $\text{sign}(\alpha) = -1$, if $\alpha < 0$. Note that the value 0 cannot appear in Algorithm `subtract`, since sign is used there only when its argument is nonzero.

Three special cases are dealt with by Algorithm `subtract`. If $M := \max(|\alpha|, |\gamma|) = 0$, then $\sigma = 0$. This case needs the absolute values of two real numbers, a max operation, a test, and setting $\phi =$ `true`. The second special case is when one of the values $|\alpha|$ and $|\gamma|$ is negligible compared to the other, i.e., when $m/M \leq \varepsilon/4$, with $m = \min(|\alpha|, |\gamma|)$. In such a case, $\sigma = -\gamma$ if $m = |\alpha|$, or $\sigma = \alpha$, otherwise. In addition to the operations performed for the first special case, a min operation, two divisions and two tests are needed. The third special case is when α and γ have the same sign, which implies that the result is $\sigma = \alpha - \gamma$. This case needs two sign operations, a subtraction and a test. The general case may involve few additional operations: a multiplication, an addition, one or two logical operations and a test. If the test is satisfied, then $\sigma = \text{sign}(\alpha)K$ and ϕ is set to `false`. Otherwise, $\sigma = \alpha - \gamma$.

To obtain $d_1 = |a_1 - a_2|$, Algorithm `subtract` has to be called for both the real and imaginary parts of $\sigma = a_1 - a_2 =: \sigma_r + \sigma_i\,\imath$. If it succeeded to obtain representable values for σ_r and σ_i, then $d_1 = |\sigma|$ can be obtained using Algorithm `absa`. Let $M_d = \max(|\sigma_r|, |\sigma_i|)$ be the value of M on input (and output) of Algorithm `absa` applied to σ, and let d_1 or u_d and δ_d be its outputs. If u_d is `true`, then $|a_1 - a_2| = d_1$; otherwise, M_d and δ_d are the factored representation of d_1. If the parameter ϕ is `false` on exit of the first or the second call of the algorithm, it means that σ_r or σ_i, respectively, would exceed K in magnitude and therefore d_1 could not be computed. In such a case, σ_r and/or σ_i are set to $\pm K$, or to $\pm$`Infinity`, if the IEEE arithmetic is available.

3.4. Computation of $d_2 = |1/a_1 - 1/a_2|$, Using $|a_1 - a_2|$

Algorithm `chordal` has to compute d_2 only when $a_1 \neq 0$ and $a_2 \neq 0$, since the special cases $a_1 = 0$ or $a_2 = 0$ have already been solved, resulting in $d = |a_2|$ or $d = |a_1|$, respectively. If $\phi =$ `true` on exit of both calls of Algorithm `subtract` made in Algorithm `chordal` to compute $d_1 = |a_1 - a_2|$, i.e., if d_1 is a representable result (possibly in a factored form), then $d_2 := |1/a_1 - 1/a_2|$ can be obtained

 doi:10.20944/preprints202505.1817.v1

more efficiently than using the direct formula, based on evaluating $1/a_1 - 1/a_2$, as done in Section 3.5. Indeed, since $1/a_1 - 1/a_2 = (a_2 - a_1)/(a_1 a_2)$, it follows that

$$d_2 = \left| \frac{1}{a_1} - \frac{1}{a_2} \right| = \frac{|a_2 - a_1|}{|a_1||a_2|} = \frac{d_1}{|a_1||a_2|}. \tag{5}$$

The absolute values of a_1 and a_2 are easily computed using Algorithm `absa`. Then, d_2 can be obtained using (5), taking into account that all factors, d_1, $|a_1|$, and $|a_2|$ can be in factored form or not. From (5), it follows that $d_2 > d_1$ if $|a_1||a_2| < 1$, hence d_2 should not be evaluated in this case, since then $d = d_1$. The computations are summarized in Algorithm `d2byd1` shown below.

The order of the arithmetic operations in Algorithm `d2byd1` is important; changing it could produce overflows or underflows. Parentheses are used to enforce that order, and avoid possible optimizations made by compilers. Note that if $\neg u_1 \vee \neg u_2$, i.e., in the two `else` cases of the test branches for u_d, then M and m are already available from Algorithm `chordal`. Another observation is that if $u_1 = u_2 = u_d =$ `true` and $M = \max(|a_1|, |a_2|) < 1$, it follows that $d_2 > d_1$, since $d_2 = (d_1/M)/\min(|a_1|, |a_2|)$, where also $\min(|a_1|, |a_2|) \leq M < 1$; therefore $d = d_1$. Note that $|a_1||a_2| > 1$ in the other three combinations of u_1 and u_2 for $u_d =$ `true`, since either $|a_1|$ or $|a_2|$ (or both) are larger than K, and the other value must be at least $K\varepsilon/4$, because otherwise the special case $f =$ `true`, already detected by Algorithm chordal, appears.

If $\phi =$ `false` in Algorithm `subtract`, d_2 cannot be computed by Algorithm `d2byd1` since d_1 is not available, and therefore d_2 has to be found using the direct formula, which involves more computations.

Algorithm 4 `d2byd1` computes d_2 in (1) using d_1

Require: $a_1, a_2 \in \overline{\mathbb{C}}_r, M_1, u_1, M_2, u_2, d_1$ (or M_d, δ_d), u_d, M, m, K
Ensure: d_2 (or d, M, if $u_1 = u_2 = u_d =$ `true` and $M < 1$)
  $[u_1, \mu_1, \delta_1] =$ `absa`(a_1, M_1, u_1, K), $[u_2, \mu_2, \delta_2] =$ `absa`(a_2, M_2, u_2, K)
  if u_d **then**
    if $u_1 \wedge u_2$ **then**
      $M = \max(\mu_1, \mu_2)$
      if $M < 1$ **then**
        $d = d_1$
      else
        $d_2 = (d_1/M)/\min(\mu_1, \mu_2)$
      end if
    else if u_1 **then**
      $d_2 = \big((d_1/\delta_2)/M_2\big)/\mu_1$
    else if u_2 **then**
      $d_2 = \big((d_1/\delta_1)/M_1\big)/\mu_2$
    else
      $d_2 = \big((d_1/\delta_1/\delta_2)/M\big)/m$
    end if
  else
    if $u_1 \wedge u_2$ **then**
      $d_2 = \big(\delta_d/\min(\mu_1, \mu_2)\big)\big(M_d/\max(\mu_1, \mu_2)\big)$
    else if u_1 **then**
      $d_2 = \big((\delta_d/\delta_2)/\mu_1\big)\big(M_d/M_2\big)$
    else if u_2 **then**
      $d_2 = \big((\delta_d/\delta_1)/\mu_2\big)\big(M_d/M_1\big)$
    else
      $d_2 = \big((\delta_d/\delta_1/\delta_2)/m\big)\big(M_d/M\big)$
    end if
  end if

 doi:10.20944/preprints202505.1817.v1

Besides the evaluations of $|a_1|$ and $|a_2|$ or of their factored representations, in the usual case, with $u_d = \texttt{true}$, Algorithm `d2byd1` needs three, four or four tests, and three, three or four divisions, for the cases when only $u_1 = \texttt{true}$ or $u_2 = \texttt{true}$, or when $u_1 = u_2 = \texttt{false}$, respectively; the case $u_1 = u_2 = \texttt{true}$ needs three tests, one $\wedge$, one max and one min operations, and two divisions. If $u_d = \texttt{false}$, an additional multiplication with M_d is also required in all these four cases.

3.5. Direct Computation of $d_2 := |1/a_1 - 1/a_2|$

The direct computation of d_2 using its definition in (1),

$$d_2 = |1/a_1 - 1/a_2|, \tag{6}$$

needs to evaluate the reciprocals of a_1 and a_2, then the difference $1/a_1 - 1/a_2$, and finally the absolute value of this difference. The following algorithm shows how the reciprocal of a complex number a has to be safely computed. Note that if $a = \alpha + \beta\imath$, $1/a = (\alpha - \beta\imath)/|a|^2$, where $|a|$ can be given in a factored form, $M\delta$, but this product might not be representable. However, squaring up $|a|$ is prone to overflow, and therefore it must be avoided. Algorithm `inva` listed below shows how $1/a$ should be evaluated.

Algorithm 5 `inva` computes the reciprocal of a complex number given its modulus or factored representation

Require: $a = \alpha + \beta\imath \in \overline{\mathbb{C}}_r$, $|a|$ if $u = \texttt{true}$, or M, δ, if $u = \texttt{false}$, K
Ensure: $1/a = \mu + \nu\imath$
  if u **then**
    $\mu = (\alpha/|a|)/|a|$, $\nu = -(\beta/|a|)/|a|$
  else
    $\mu = \Big(((\alpha/\delta)/\delta)/M\Big)/M$, $\nu = -\Big(((\beta/\delta)/\delta)/M\Big)/M$
  end if

Remark 4. *The minus sign in front of ν is not necessary in this context, since the real and imaginary parts will be used to evaluate $|1/a_1 - 1/a_2|$ using Algorithm* `absa`*. Indeed, the signs of the imaginary parts of $1/a_1$ and $1/a_2$ do not matter.*

Algorithm `inva` needs one test and four divisions if $u = \texttt{true}$, or six divisions if $u = \texttt{false}$, but it should be applied to both $1/a_1$ and $1/a_2$. Therefore, two tests and eight, ten or twelve divisions are necessary if $u_1 = u_2 = \texttt{true}$, $u_i = \texttt{true}$ and $u_j = \texttt{false}$, for $i, j \in \{1, 2\}$, $i \neq j$, or $u_1 = u_2 = \texttt{false}$, respectively. The remaining computations are the same as those for computing d_1, but applied to $|1/a_1 - 1/a_2|$. Therefore, the computational effort for computing $d_2 = |1/a_1 - 1/a_2|$ using Algorithm `inva` shoud be compared to that of Algorithm `d2byd1`, which needs at most four tests and four divisions, or at most three tests, two divisions, one $\wedge$, one max and one min operations, if $u_1 = u_2 = u_d = \texttt{true}$; if $u_d = \texttt{true}$, another multiplication is needed.

Clearly, the direct approach needs more computations than the first approach, based on d_1. Note that the first approach can easily and efficiently detect the case when $|1/a_1 - 1/a_2|$ is not needed, since its value will exceed $|a_1 - a_2|$. However, the direct approach has to be used when the first approach could not be used, namely, when d_1 overflows.

Remark 5. *The implementation inlines the low level algorithms, except for Algorithm* `subtract`*, that is more involved; it is called two times, if d_2 can be found using d_1, or four times, otherwise. Similarly, Algorithm* `inva` *is either not called, or called twice. The very simple Algorithm* `absa` *is called at most four times, and Algorithm* `d2byd1` *is inlined once.*

 doi:10.20944/preprints202505.1817.v1

4. Numerical Results

This section starts by presenting four examples that show the significance of the proposed approach. Then, the numerical results obtained in a large experiment with randomly generated examples, which cover the entire range of representable numbers, are discussed. Finally, the performance of two solution approaches for a block diagonalization problem of order 999 is analyzed, illustrating the advantages of using a good eigenvalue reordering strategy and fast algorithms for computing the approximate symmetric chordal metric.

4.1. Detailed Examples

The numerical results for four examples are presented and compared with those got by the direct use of MATLAB computations.

Example 1. *Let $a_1 = K + K/10\,\imath$, $a_2 = K/10 + K\,\imath$. Implementing formula (1) in MATLAB, the result is $d = 0$, while Algorithm* `chordal`*, together to the lower level algorithms, returns* $\underline{d}$ = `7.010041250456554e-309`*. Both d_1 and d_2 computed using (1) in MATLAB are wrong. Specifically, $a_1 - a_2 = \sigma - \sigma\imath$, with $\sigma =$* `1.617923821376084e+308`*, and the MATLAB command* `abs`$(a_1 - a_2)$ *gives $d_1 =$* `Inf`*. Moreover, evaluating $1/a_i$, it follows that $1/a_1 = 1/a_2 = 0$, and therefore $d = d_2 = 0$. On the other hand, the factored form of $|a_1 - a_2|$ has $M_d = \sigma$ and $\delta_d = \sqrt{2}$. Moreover, $M = K$ and $m = K/10$, so that $\delta_1 = \delta_2 =: \delta = \sqrt{1 + (m/M)^2} =$* `1.004987562112089`*; therefore, $|a_1 - a_2|/|a_1|/|a_2| = \big((\delta_d/\delta/\delta)/M\big)\big(M_d/M\big) = \underline{d} > 0$.*

Example 2. *Let $a =$* `1.16e308` $+$ `1.66e308`$\imath$*. MATLAB command* `1/a` *gives the result 0. While this is close to the true value, it is qualitatively wrong, since it does not satisfy the mathematical condition that $a \times (1/a) = 1$. But using the factored form of $|a|$, $|a| = M\delta$, with $M =$* `1.66e308` *and $\delta = \sqrt{1 + (m/M)^2} =$* `1.219965042368649`*, as $m =$* `1.16e308`*, it follows that*

$$\begin{aligned} 1/a = \bar{a}/|a|^2 &= (\texttt{1.16e308} - \texttt{1.66e308}\,\imath)/M/\delta/M/\delta \\ &= \texttt{2.828440456451768e-309} - \texttt{4.047595825612014e-309}\,\imath, \end{aligned} \tag{7}$$

and then $a \times \big(1/a\big) =$ `9.999999999999996e-01` $-$ `1.665334536937735e-16`$\imath$*, which is very close to 1; the error is of the order of ε. Note that the denominator in (7) should be evaluated as shown there (or with permuted factors), but not as $|a|^2$; multiplying $M\delta$ would give* `Inf`*. Clearly, the result obtained is very close to 0, but it can be accurately obtained and it numerically verifies the condition on reciprocal of a complex number, while 0 does not. Note also that $|1/a|$ can be obtained directly as $1/M/\delta$.*

For computing d_2 in (1), it is necessary to evaluate both $1/a_1$ and $1/a_2$, and then the absolute value of their difference. If a_1 and/or a_2 have real and imaginary parts with very big magnitude, as in Example 2, then using (1) will give an inaccurate result for d, as shown in the next example.

Example 3. *Let $a_1 = a$ from Example 2, and $a_2 = K + K\imath$. Implementing (1) in MATLAB, the result is $d =$* `3.933412034978399e-309`*, while the implementation of Algorithm* `chordal` *gives* $\underline{d} =$ `1.267129104195721e-309`*. The absolute difference between these values is very small, about* `2.66628e-309`*. However, their relative difference, $|d - \underline{d}|/d$ is about 0.67785, and taking $\underline{d}$ as reference, $|d - \underline{d}|/\underline{d} \approx 2.10419$. These big relative differences are due to the fact that using (1) to evaluate $d = d_2$ in MATLAB gives $1/a_1 = 0$, and $d_2 = |1/a_2| = 1/K/\sqrt{2} = d$. Both MATLAB and Algorithm* `chordal` *compute $d_1 =$* `6.523894033771084e+307`*, but Algorithm* `chordal` *uses Algorihm* `d2byd1` *to obtain $\underline{d} = d_2 = d_1/|a_1|/|a_2|$, where the factored representations are used for $|a_1|$ and $|a_2|$.*

Example 4. *Let $a = K/(4/3) + K/2\,\imath$. Using MATLAB, it follows that* `1/a` $= 0$*, which is wrong, since it implies that $a \times (1/a) = 0$, while the true result is 1. On the contrary, Algorithm* `inva` *gives $1/a =$* `5.134785827324312e-309 - 3.423190551549538e-309`$\imath$*. With this value, it follows that $a \times (1/a) =$*

 doi:10.20944/preprints202505.1817.v1

$(1/a) \times a =$ `1.000000000000000e+00 + 3.885780586188048e-16`$\imath$, *that has absolute and relative errors less than* 2ε.

4.2. Large Experiment with Randomly Generated Examples

In a run with 4188166 random examples, the first three examples had $a_2 = 0$, $a_1 = 0$, and $a_1 = a_2 = 0$. Specifically, all examples have been generated using the following MATLAB commands, where `a1` and `a2` are used for a_1 and a_2, respectively.

```
a1 = randn + 1i*randn; a2 = 0;
a2 = a1; a1 = 0;
a2 = 0;
for ii = -1022 : 1023
    s1 = 2^ii;  re = s1*randn;  im = s1*randn;
    if abs( re ) > realmax,  re = sign( re )*realmax;  end
    if abs( im ) > realmax,  im = sign( im )*realmax;  end
    a1 = re + 1i*im;
    for jj = -1022 : 1023
        s2 = 2^jj;  re = s2*randn;  im = s2*randn;
        if abs( re ) > realmax,  re = sign( re )*realmax;  end
        if abs( im ) > realmax,  im = sign( im )*realmax;  end
        a2 = re + 1i*im;
    end
end
s1 = realmax;  re = s1*randn;  im = s1*randn;
if abs( re ) > realmax,  re = sign( re )*realmax;  end
if abs( im ) > realmax,  im = sign( im )*realmax;  end
a1 = re + 1i*im;
for jj = -1022 : 1023
    s2 = 2^jj;  re = s2*randn;  im = s2*randn;
    if abs( re ) > realmax,  re = sign( re )*realmax;  end
    if abs( im ) > realmax,  im = sign( im )*realmax;  end
    a2 = re + 1i*im;
end
s2 = realmax;  re = s2*randn;  im = s2*randn;
if abs( re ) > realmax,  re = sign( re )*realmax;  end
if abs( im ) > realmax,  im = sign( im )*realmax;  end
a2 = re + 1i*im;
```

The `for` loop with counter `ii` generates numbers `s1` between $k =$ `2.225073858507201e-308` (`realmin`) and `8.988465674311580e+307`, which is very close to, but smaller than $K/2$, $K =$ `1.797693134862316e+308` (`realmax`), the relative error being smaller than `5.552e-17`. The number `s1` multiplies pseudorandom double precision values drawn from the standard normal distribution (with mean 0 and standard deviation 1), to obtain the real and imaginary parts of a_1. If any of the computed parts of a_1 exceeds K in magnitude, i.e., if $\pm$`Inf` is obtained, that value is reset to $\pm K$. The number a_2 is obtained in the same way in the internal loop with counter `jj`. The second part of the code segment has `s1` $= K$ and use it to obtain a_1; a_2 is generated as above. The last part also sets `s2` $= K$, and use it to obtain a_2. Clearly, complex numbers with $|a_1| > K$ and/or $|a_2| > K$ are generated.

For reproductibility of the results, the command sequence above has been run after the MATLAB statement

```
rng( 'default' )
```

 doi:10.20944/preprints202505.1817.v1

which ensures that the (initial) seed of the random number generator is the same, and therefore the same sequence of random numbers is generated after using this command. However, for some further tests, the generating sequence has been run repeateadly several times, but the `rng( 'default' )` command has been placed just before the first run. This way, a larger number of tests have been performed.

Inside the (internal) loops, an implementation of Algorithm `chordal` is called, and its results are compared with those obtained using a MATLAB command for (1), but also a more sophisticated MATLAB code based on algorithms described in Section 3. This is needed since using (1) directly might return inaccurate results, as shown in Section 4.1.

The maximum relative error in this run has been `6.3088e-16`; the relative error is defined as $|d - \underline{d}|/\max(1, d)$, where d and $\underline{d}$ denote the results obtained by evaluating (1) in MATLAB and by using an implementation of Algorithm `chordal`, respectively. If the absolute error is computed as $|d - \underline{d}|/d$, if $d \neq 0$, and $|d - \underline{d}|$, otherwise, its maximum has been `0.2649`. The value became zero after a correction using a MATLAB code based on algorithms described in the previous section. After ten consecutive runs (without reinitialization of the random number generator), the maximum relative error and its mean value have been `7.6144e-16` and `6.3663e-16`, respectively, while using the second definition, the values have been `0.73077` and `0.35080`, and they became zero after correction.

In this test sequence, there were 3970109 examples with a_1 or a_2 negligible compared to the other, hence the logical variable f in Algorithm chordal was `true`. This represents 94.794% from the total number of examples. Specifically, 1985994 and 1984115 examples have $M = M_1$ and $M = M_2$, respectively. From those with $M = M_1$, 987937 have $d = |a_1|$, 994601 have $d = 1/|a_2|$, and 3456 have $d = \min\left(|a_1|, 1/|a_2|\right)$. From examples with $M = M_2$, 987834 have $d = |a_2|$, 992853 have $d = 1/|a_1|$, and 3428 have $d = \min\left(|a_2|, 1/|a_1|\right)$. These categories correspond to the three cases, with $m < 0.1/M$, $m > 1.1/M$, and $0.1/M \leq m \leq 1.1/M$, respectively. For $M = M_i$, the logical variables u_i and u_j, $j \neq i$, have been `true` for all examples satisfying $m < 0.1/M$ and $m > 1.1/M$, respectively. Moreover, for examples satisfying $0.1/M \leq m \leq 1.1/M$, when $|a_1|$ as well as $|a_2|$ needed to be computed, both u_1 and u_2 have been `true`. Using a factored representation has never been necessary for any of these 3970109 examples.

For the remaining 218055 examples, representing 5.206% of all examples, a_1 and a_2 are nonzero, distinct and relatively "close" to each other, since $m/M > \varepsilon/4$. Algorithm `subtract` has been used to compute the real and imaginary parts, σ_r and σ_i, of $\sigma := a_1 - a_2$. For three and two examples, σ_r or σ_i, respectively, would exceed K in magnitude. For these five examples with $\phi =$ `false`, d_1 cannot be computed, hence $d = d_2$, and consequently, $|1/a_1 - 1/a_2|$ needed to be directly computed calling twice Algorithm `inva` (for $1/a_1$ and $1/a_2$) and Algorithm `subtract` (for the real and imaginary parts of $1/a_1 - 1/a_2$). For the other 218050 examples, d_1 has been computed as $|\sigma|$, using Algorithm `absa`, and d_1 could be used to obtain d_2 via Algorithm `d2byd1`. Specifically, u_d was `true` on input to `absa`, called for evaluating d_1 for 217980 examples, and it was `false` for the remaining 70 examples with expected overflow; but u_d could be reset to `true` for 12 examples, for which d_1 could be evaluated as $d_1 = M_d\delta_d$; therefore, d_1 needed a factored representation, defined by M_d and δ_d, for only 58 examples.

All these 218055 examples needed both $|a_1|$ and $|a_2|$ values. A factored representation for $|a_1|$ has been required for 56 examples. Although 21 examples were expected to require a factored representation for $|a_2|$, the factors could be multiplied without overflows for nine of these examples.

Since d_1 could not be computed for five examples, due to expected overflows in σ_r or σ_i, it follows that d_2 can be found using Algorithm `d2byd1` for 218050 examples. But for 107475 of them, with $u_1 \wedge u_2 =$ `true`, d_2 was not computed, since $\max\left(|a_1|, |a_2|\right) < 1$, which implies that $d_2 > d_1$, hence $d = d_1$. For 110517 examples, d_2 has been evaluated using d_1, $|a_1|$ (or M_1 and δ_1) and $|a_2|$ (or M_2 and δ_2); therefore, for these examples, $d = \min\left(d_1, d_2\right)$. Specifically, the case $u_1 \wedge u_2 =$ `true`, when $d_2 = d_1/|a_1|/|a_2|$, happened for 110514 examples; the case $u_1 =$ `false` and $u_2 =$ `true`, when $d_2 = \left(\left(d_1/\delta_1\right)/M_1\right)/|a_2|$, and the case $u_1 \wedge u_2 =$ `false`, when $d_2 = \left(\left(d_1/\delta_1/\delta_2\right)/M\right)/m$, appeared for just one and two examples, respectively. There was no case with $u_1 =$ `true` and $u_2 =$ `false`.

 doi:10.20944/preprints202505.1817.v1

For the remaining 58 examples for which d_1 needed a factored representation, defined by M_d and δ_d, there were six cases with $u_1 = \texttt{true}$, 51 cases with $u_2 = \texttt{true}$ and one case with $u_1 \wedge u_2 = \texttt{false}$. The corresponding formulas are $d_2 = \left(\left(\delta_d/\delta_j\right)/M_i\right)\left(M_\delta/M_j\right)$, for $j \neq i$, $i = 1, 2$, and $d_2 = \left(\left(\delta_d/\delta_1/\delta_2\right)/m\right)\left(M_\delta/M\right)$, respectively. The case with $u_1 \wedge u_2 = \texttt{true}$ never appeared; for it, the formula would be $d_2 = \left(\delta_d/\min\left(|a_1|, |a_2|\right)\right)\left(M_\delta/M\max\left(|a_1|, |a_2|\right)\right)$, for $u_1 \wedge u_2 = \texttt{false}$.

Finally, for the five examples for which d_1 could not be computed, since σ_r or σ_i exceeded K, $d = d_2$ has been evaluated by calling Algorithm `inva` twice, for a_1 and a_2. The logical variables u_1 and u_2 were `true` for four and one examples, respectively, and `false` for one and four examples. Therefore all formulas for μ and ν in Algorithm `inva` have been applied. Then, $\tau = 1/a_1 - 1/a_2$ has been obtained using Algorithm `subtract` separately for the real and imaginary parts of τ; the variable ϕ remained `true` for each of these examples. The result $d_2 = |\tau|$ has been found by Algorithm `absa`.

These results show that all examples have been solved using the simplest possible formulas, which prove the efficiency of the implementation. Moreover, it was checked out that the results have been practically the same with those obtained when d_2 has been always computed using (6). This demonstrates that the accuracy of the results was preserved using the simplified formulas.

4.3. Block Diagonalization of a Matrix Pencil

This subsection summarizes the results obtained for block diagonalization of a matrix pencil of order $n = 999$, with 107 real and 446 complex conjugate eigenvalues, revealing dense clusters. The numerical approach has been briefly discussed in Section 1. Without using clustering information, a solution with 135 diagonal block pairs has been computed in 62.68 s (seconds). The largest block pair, of order 734, appeared in the 89-th block position on the diagonals; in addition, there are three 1×1 and 131 2×2 diagonal block pairs. The remaining 104 real eigenvalues are included in the largest block pair.

Using an eigenvalue reordering strategy based on pairwise "distances" between eigenvalues, measured by the approximate symmetric chordal metric, another solution has been computed in 4.6905 s, hence the execution time has been 13.36 times shorter. More block pairs, specifically 141, have been found and the largest block pair, of order 719, appeared in the last position. This is the preferred situation, since well separated eigenvalues could be quickly placed in small block pairs in the leading part, while big clusters remain to be placed in the trailing part of the matrix pencil. There were 140 2×2 block pairs, and so, all real eigenvalues were available in the largest block pair. The efficiency gain is mainly due to a better eigenvalue reordering, which reduced the number of failed attempts to solve generalized Lyapunov equations. However, it is worth to mention that the number of "distances" between eigenvalues is $n(n+1)/2$, hence it increases quadratically with the problem size. Therefore, it is important that the algorithms for computing the approximate symmetric chordal metric be as fast as possible.

5. Conclusions

The basic theoretical properties of the approximate symmetric chordal metric for two real or complex numbers are investigated, and reliable, accurate and efficient algorithms for its computation are proposed. A factored representation of the modulus of a complex number is introduced, which allows to obtain very accurate results for the entire range of the floating point number system of a computer. Two different ways to evaluate the distance between the reciprocals of the given numbers are described. The algorithms can be easily implemented on various architectures and compilers. Extensive numerical tests have been performed to assess the performance of the associated implementation. The results have been compared to those obtained in MATLAB, but with appropriate modifications for numbers very close to the bounds of the range of representable values, where usual formulas give wrong results, as shown in several detailed numerical examples. A block diagonalization example is also discussed to illustrate the efficiency improvement possible using a proper eigenvalue reordering strategy and fast algorithms for computing the approximate symmetric chordal metric.

 doi:10.20944/preprints202505.1817.v1

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Klamkin, M.S.; Meir, A. Ptolemy's inequality, chordal metric, multiplicative metric. *Pac. J. Math.* **1982**, *101*, 389–392. https://doi.org/10.2140/pjm.1982.101.389.
2. Sasane, A. A generalized chordal metric in control theory making strong stabilizability a robust property. *Complex Anal. Oper. Theory* **2013**, *7*, 1345–1356. https://doi.org/10.1007/s11785-012-0239-5.
3. Papadimitrakis, M. *Notes on Complex Analysis*. 2019. [Online]. Available: http://fourier.math.uoc.gr/~papadim/complex_analysis_2019/gca_vvn_r.pdf. [Accessed: 3-May-2025].
4. Rainio, O.; Vuorinen, M. Lipschitz constants and quadruple symmetrization by Möbius transformations. *Complex Anal. Synerg.* **2024**, *20*, 8 pp. https://doi.org/10.1007/s40627-024-00136-y.
5. Bishop, C. *Complex Analysis I, MAT 536, Spring 2024*. Stony Brook University, Dep Mathematics, 100 Nicolls Road, Stony Brook, NY 11794, 2024.
6. Álvarez Tuñón, O.; Brodskiy, Y.; Kayacan, E. Loss it right: Euclidean and Riemannian metrics in learning-based visual odometry. *arXiv:2401.05396v1 [cs.CV]* **2023**. https://doi.org/10.48550/arXiv.2401.05396.
7. Xu, W.; Pang, H.K.; Li, W.; Huang, X.; Guo, W. On the explicit expression of chordal metric between generalized singular values of Grassmann matrix pairs with applications. *SIAM J. Matrix Anal. Appl.* **2018**, *39*, 1547–1563. https://doi.org/10.1137/17M1140510.
8. Sun, J.-g. Perturbation theorems for generalized singular values. *J. Comput. Math.* **2025**, *1*, 233–242.
9. Sasane, A. A refinement of the generalized chordal distance. *SIAM J. Control & Optimiz.* **2014**, *52*, 3538–3555. https://doi.org/10.1137/130918095.
10. Bavely, C.; Stewart, G.W. An algorithm for computing reducing subspaces by block diagonalization. *SIAM J. Numer. Anal.* **1979**, *16*, 359–367. https://doi.org/10.1137/0716028.
11. Golub, G.H.; Van Loan, C.F. *Matrix Computations*, fourth ed.; The Johns Hopkins University Press: Baltimore, Maryland, 2013.
12. Wilkinson, J.H. *The Algebraic Eigenvalue Problem*; Oxford University Press (Clarendon): Oxford, UK, 1965.
13. Wilkinson, J.H. Kronecker's canonical form and the QZ algorithm. *Lin. Alg. Appl.* **1979**, *28*, 285–303.
14. Stewart, G.W. On the sensitivity of the eigenvalue problem $Ax = \lambda Bx$. *SIAM J. Numer. Anal.* **1972**, *9*, 669–686. https://doi.org/55510.1137/0709056.
15. Demmel, J.W. The condition number of equivalence transformations that block diagonalize matrix pencils. *SIAM J. Numer. Anal.* **1983**, *20*, 599–610.
16. Kågström, B.; Westin, L. Generalized Schur methods with condition estimators for solving the generalized Sylvester equation. *IEEE Trans. Automat. Contr.* **1989**, *34*, 745–751. https://doi.org/10.1109/9.29404.
17. Anderson, E.; Bai, Z.; Bischof, C.; Blackford, S.; Demmel, J.; Dongarra, J.; Du Croz, J.; Greenbaum, A.; Hammarling, S.; McKenney, A.; et al. *LAPACK Users' Guide: Third Edition*; Software · Environments · Tools, SIAM: Philadelphia, 1999.
18. MathWorks®. MATLAB™, *Release R2024b*. The MathWorks, Inc., 3 Apple Hill Drive, Natick, MA, 01760, 2024.
19. Wolfram, S. The Story Continues: Announcing Version 14 of Wolfram Language and Mathematica, 2024. [Online]. Available: http://writings.stephenwolfram.com/2024/01/the-story-continues-announcing-version-14-of-wolfram-language-and-mathematica. [Accessed: 3-May-2025].
20. Malcolm, M.A. Algorithms to reveal properties of floating-point arithmetic. *Commun. ACM* **1972**, *15*, 949–951.
21. Gentleman, W.M.; Marovich, S.B. More on algorithms that reveal properties of floating point arithmetic units. *Commun. ACM* **1974**, *17*, 276–277. https://doi.org/10.1145/360980.361003.
22. Boyd, S.; Vandenberghe, L. *Convex Optimization*; Cambridge University Press: Cambridge, United Kingdom, 2004.