
NUMERICALLY STABLE EVALUATION OF CLOSED-FORM EXPRESSIONS FOR EIGENVALUES OF 3×3 MATRICES

PREPRINT

Michal HABERA
Department of Engineering
University of Luxembourg
michal.habera@uni.lu

Andreas ZILIAN
Department of Engineering
University of Luxembourg
andreas.zilian@uni.lu

October 2025

ABSTRACT

Trigonometric formulas for eigenvalues of 3×3 matrices that build on Cardano's and Viète's work on algebraic solutions of the cubic are numerically unstable for matrices with repeated eigenvalues. This work presents numerically stable, closed-form evaluation of eigenvalues of real, diagonalizable 3×3 matrices via four invariants: the trace I_1 , the deviatoric invariants J_2 and J_3 , and the discriminant Δ . We analyze the conditioning of these invariants and derive tight forward error bounds. For J_2 we propose an algorithm and prove its accuracy. We benchmark all invariants and the resulting eigenvalue formulas, relating observed forward errors to the derived bounds. In particular, we show that, for the special case of matrices with a well-conditioned eigenbasis, the newly proposed algorithms have errors within the forward stability bounds. Performance benchmarks show that the proposed algorithm is approximately ten times faster than the highly optimized LAPACK library for a challenging test case, while maintaining comparable accuracy.

Keywords numerically stable eigenvalues, 3×3 matrices, closed-form expressions, polynomial roots, numerical stability

1 Introduction and motivation

The classical textbook formulas for closed-form expressions of eigenvalues of a diagonalizable matrix $\mathbf{A} \in \mathbb{R}^{3 \times 3}$ with real spectrum are based on the trace of the matrix I_1 , and two deviatoric matrix invariants J_2 and J_3 ,

$$\begin{aligned} I_1(\mathbf{A}) &:= \text{tr}(\mathbf{A}), \\ J_2(\mathbf{A}) &:= -\frac{1}{2} \left[\text{tr}(\text{dev}(\mathbf{A}))^2 - \text{tr}(\text{dev}(\mathbf{A})^2) \right] = \frac{1}{2} \text{tr}(\text{dev}(\mathbf{A})^2), \\ J_3(\mathbf{A}) &:= \det(\text{dev}(\mathbf{A})). \end{aligned} \tag{1}$$

The three eigenvalues λ_k are then given by (see Smith, 1961 or Bronshtein et al., 2015, §1.6.2.3 or Press et al., 2007, Eq. 5.6.12),

$$\lambda_k = \frac{1}{3} \left(I_1 + 2\sqrt{3J_2} \cos \left(\frac{\varphi + 2\pi k}{3} \right) \right), \quad k \in \{1, 2, 3\}, \tag{2}$$

where the triple-angle φ is computed as

$$\varphi := \arccos \left(\frac{3\sqrt{3}}{2} \frac{J_3}{J_2^{3/2}} \right). \tag{3}$$

The above expressions are notoriously unstable in finite-precision arithmetic, especially when eigenvalues coalesce. A typical pitfall of closed-form approaches is the reduction of the eigenvalue problem to the computation of roots of a cubic polynomial, see Fig. 1. This approach, i.e., the computation of the roots of a cubic polynomial given its monomial

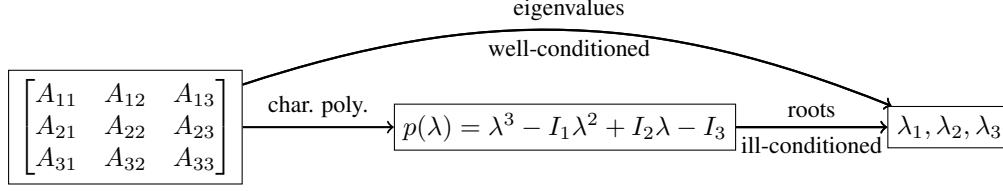


Figure 1: Typical approach for computing eigenvalues of 3×3 matrices via characteristic polynomial and its roots.

coefficients, is known to be ill-conditioned, see (Trefethen and Bau, 1997, p. 110) and (Higham, 2002, §26.3.3.). While we do not bypass the utilization of the characteristic polynomial, we try to improve the numerical stability of the overall process by improving the stability of the individual steps, and potentially computing additional, seemingly redundant invariants that help stabilize the computation of the eigenvalues.

According to (Blinn, 2007), the first known approach for improving the numerical stability is from (La Porte, 1973) who proposed to use the identity $\tan(\arccos x) = \sqrt{1 - x^2}/x$ in the context of solving roots of a general cubic polynomial. When applied to the matrix eigenvalue problem, the triple-angle expression takes the form

$$\varphi = \arctan \left(\frac{\sqrt{27(4J_2^3 - 27J_3^2)}}{27J_3} \right) = \arctan \left(\frac{\sqrt{27\Delta}}{27J_3} \right) \quad (4)$$

making use of the matrix *discriminant* $\Delta := 4J_2^3 - 27J_3^2$. Eq. (4) has the advantage of evaluating the $\arctan(x) = x - x^3/3 + \mathcal{O}(x^5)$ around zero (for matrices with repeated eigenvalues), which is numerically more stable than evaluating the $\arccos(x)$ around one.

Another notable improvement is based on the work of (Scherzinger and Dohrmann, 2008), who proposed an algorithm for *symmetric* 3×3 matrices based on computing the distinct eigenvalue first, then deflating the matrix to a 2×2 problem for which Wilkinson’s shift is used to compute the remaining eigenvalues. This approach is stable for symmetric matrices, but it does not generalize to nonsymmetric matrices. In addition, it is not a closed-form expression and requires branching and conditional statements.

The computation of the matrix discriminant Δ itself is also prone to numerical instability, as it involves subtraction of two potentially close quantities, $4J_2^3$ and $27J_3^2$. The first work addressing this issue in the context of 3×3 matrices is (Habera and Zilian, 2021) and is based on the factorization of the discriminant from (Parlett, 2002) into a sum of products of terms that vanish as the matrix approaches a matrix with multiple eigenvalues.

Recently, an alternative factorization of the discriminant Δ for symmetric matrices based on the Cayley–Hamilton theorem was proposed in (Harari and Albocher, 2022). The authors then published a follow-up paper (Harari and Albocher, 2023) where the Cayley–Hamilton factorization is abandoned in favor of a simpler *sum-of-squares* formula for the discriminant.

In (Habera and Zilian, 2021) we advocated replacing the traditional discriminant expressions with *sum-of-products* or *sum-of-squares* formulas that avoid catastrophic cancellation. Unfortunately, as discussed in (Habera and Zilian, 2021), the proposed algorithm failed to achieve eigenvalues with satisfactory accuracy for matrices with $J_2 \rightarrow 0$. In addition, the benchmarks and interpretation of errors were intuitive, but lacked rigorous forward or backward error analysis. We used scaled invariants $\Delta_p = 3J_2$ and $\Delta_q = 27J_3$. In the present work, we use the classical definitions of the invariants for consistency with the existing literature, especially in the engineering community where J_2 and J_3 are widely used in constitutive modeling of materials. In addition, we improve the numerical stability in the limit case $J_2 \rightarrow 0$ by proposing improved algorithms for the computation of J_2 , J_3 , and Δ .

The lack of error analysis is a common issue in the existing literature on closed-form expressions for eigenvalues of 3×3 matrices. Terms like “numerically stable” or “robust” are often used without rigorous justification or derivation of error bounds. We address this gap. Additionally, only in (Habera and Zilian, 2021) and this work is the numerical stability for the *generalized case of nonsymmetric matrices* considered.

On the other hand, the typical approach to computing eigenvalues of general matrices uses iterative algorithms, such as the QR algorithm, which are implemented in standard libraries like LAPACK (Anderson et al., 1999). These algorithms are based on numerically stable orthogonal transformations to reduce the matrix to a simpler form (e.g., Hessenberg form) and then iteratively applying the QR algorithm to converge to the eigenvalues. Unsurprisingly, these iterative algorithms are routinely used in practice even for small 3×3 matrices.

Despite the widespread use of iterative algorithms, closed-form expressions for eigenvalues remain important due to several reasons: 1. number of floating-point operations is significantly lower than for iterative algorithms, which is critical in performance-sensitive applications, and 2. they allow for symbolic differentiation, which is important when sensitivities or gradients are required, e.g., in optimization or machine learning applications. The latter was explored in (Habera and Zilian, 2021), where the relation

$$\mathbf{E}_k^\top = \frac{\partial \lambda_k}{\partial \mathbf{A}} \quad (5)$$

was used to compute eigenprojectors (i.e., matrices projecting onto the eigenspaces associated with the eigenvalues λ_k). With the eigenprojectors available in closed-form, one can compute functions of matrices (e.g., the matrix exponential) and their derivatives in closed-form as well. In addition, in the case of a matrix parametrized by some variable $t \in \mathbb{R}$, i.e., $\mathbf{A}(t) : t \mapsto \mathbf{A}(t)$ one can use the closed-form expressions and their derivatives to study the analytical dependence of eigenvalues and eigenvectors on the parameter t . Lastly, the use of trigonometric solution guarantees that the eigenvalues are ordered $\lambda_1 \leq \lambda_2 \leq \lambda_3$, which is not the case for iterative algorithms. Ordering of eigenvalues is important in many applications, e.g., in engineering mechanics when computing principal stresses or strains.

2 Numerical stability

In this work, we use the notation and definitions from (Higham, 2002) and (Trefethen and Bau, 1997). We follow the standard IEEE 754 model with

$$\text{fl}(x \text{ op } y) = (x \text{ op } y) (1 + \delta), \quad \text{op} \in \{+, -, *, /\}. \quad (6)$$

The same applies to the floating-point representation of a number, $\text{fl}(x) = x(1 + \delta)$. The quantity δ is close to zero. More precisely, it is bounded as $|\delta| \leq \epsilon_{\text{mach}}$, where ϵ_{mach} is the unit roundoff (machine precision). In other words, each floating-point operation of type $(+, -, *, /)$ adds a relative error of at most ϵ_{mach} . For IEEE 754 double precision, we have

$$\epsilon_{\text{mach}} = \frac{1}{2} \beta^{1-t} = 2^{-53} \approx 1.11 \times 10^{-16}, \quad (7)$$

where β is the base and t is the precision (number of base- β digits).

We also use the symbol θ_n to denote the cumulative relative error of a sequence of n floating-point operations (flops), i.e.,

$$1 + \theta_n = \prod_{i=1}^n (1 \pm \delta_i)^{\pm 1}, \quad |\delta_i| \leq \epsilon_{\text{mach}}, \quad (8)$$

with the standard bound (assuming $n\epsilon_{\text{mach}} < 1$)

$$|\theta_n| \leq \frac{n\epsilon_{\text{mach}}}{1 - n\epsilon_{\text{mach}}} = \gamma_n. \quad (9)$$

An algorithm $f : V \rightarrow W$ is called *backward stable in the relative sense* if for all $x \in V$ there exists $\delta x \in V$ such that

$$\text{fl}(f(x)) = f(x + \delta x), \quad \text{where} \quad \frac{\|\delta x\|}{\|x\|} \leq C\epsilon_{\text{mach}}. \quad (10)$$

In this work, V and W are finite-dimensional vector spaces. Most often, $V = \mathbb{R}^{3 \times 3}$ and $W = \mathbb{R}$. Since we are concerned with small matrices of fixed size 3×3 , the dependence of the constant C on the problem dimension is negligible. In addition, the constant C is required to be moderate, usually $C \leq 100$, often $C \leq 10$. The symbol C will be used to denote this constant in the rest of the paper.

The quantity $\|\delta x\|/\|x\|$ is called the (relative) *backward error* of the algorithm. In other words, the algorithm is backward stable if it computes the exact result for a slightly perturbed input, where the perturbation is small relative to the input.

The *relative condition number* of a function $f : V \rightarrow W$ at x is defined as

$$\kappa_f(x) := \sup_{\delta x} \left(\frac{\|f(x + \delta x) - f(x)\|}{\|f(x)\|} \bigg/ \frac{\|\delta x\|}{\|x\|} \right), \quad (11)$$

i.e., the worst-case relative change in the output divided by a relative change in the input. Here, δx is infinitesimal. That is, the above is understood in the limit $\|\delta x\| \rightarrow 0$. For differentiable functions, the relative condition number can be expressed in terms of the Jacobian $\mathbf{J}_f(x) = \partial f / \partial x$ as (Trefethen and Bau, 1997, Eq. 12.6),

$$\kappa_f(x) = \|\mathbf{J}_f(x)\| \frac{\|x\|}{\|f(x)\|}. \quad (12)$$

The *absolute condition number* is defined as

$$\kappa_f^{\text{abs}}(x) := \sup_{\delta x} \left(\frac{\|f(x + \delta x) - f(x)\|}{\|\delta x\|} \right), \quad (13)$$

which, using the Jacobian, can be expressed as (Trefethen and Bau, 1997, Eq. 12.3),

$$\kappa_f^{\text{abs}}(x) = \|\mathbf{J}_f(x)\|. \quad (14)$$

The error of the floating-point evaluation of an algorithm f , i.e., $\|\text{fl}(f(x)) - f(x)\|$ at a point x , is called the *absolute forward error*. We say that an algorithm is *forward stable in the absolute sense* if its absolute forward error is on the order of κ_f^{abs} times the machine precision. An important result that we use throughout the paper is that the forward error is bounded by the product of the condition number and the backward error, i.e.,

$$\text{forward error} \leq \text{condition number} \cdot \text{backward error}. \quad (15)$$

The meaning of each term must be consistent, i.e., we bound absolute forward error by absolute condition number and absolute backward error, or relative forward error by relative condition number and relative backward error. Which of the two is used depends on the context and the problem at hand.

An algorithm is called *accurate* if it produces results with a small relative forward error, see (Trefethen and Bau, 1997, Eq. 14.2), i.e.,

$$\frac{\|\text{fl}(f(x)) - f(x)\|}{\|f(x)\|} \leq C \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (16)$$

Accurate algorithms produce results that are as close to the exact result as the floating-point format and machine precision allow and are the pinnacle of what one can achieve in numerical computations.

3 Benchmarks

In this section, the methodology for generating numerical benchmarks is described. It could be the case that rounding error tests are sensitive to the specific libraries, compilers, and hardware used. We describe the procedures in detail to allow reproducibility. We also provide the data and code used to generate the results in this paper as part of open-source library `eig3x3`, see (Habera and Zilian, 2025).

Algorithms for evaluating the invariants in IEEE 754 double-precision floating-point were implemented in C11 with Python wrappers via CFFI (Rigo and Fijalkowski, 2025) using the `double` 64-bit floating-point format and in NumPy 2.3.4 (Harris et al., 2020) using the `numpy.float64` data type. In order to compute the forward error of a function $f(x)$, we compute the reference value $f_{\text{ref}}(x)$ using the `mpmath` 1.3.0 library (Fredrik Johansson, 2023), with precision set to a high number of decimal places, i.e., `mpmath.dps = 256`.

In order to capture several limit cases of the eigenvalue multiplicities and conditioning of the eigenvectors, we consider test input matrices computed as

$$\text{fl}(\mathbf{A}) = \text{fl}(\mathbf{U}\mathbf{D}\mathbf{U}^{-1}) \quad (17)$$

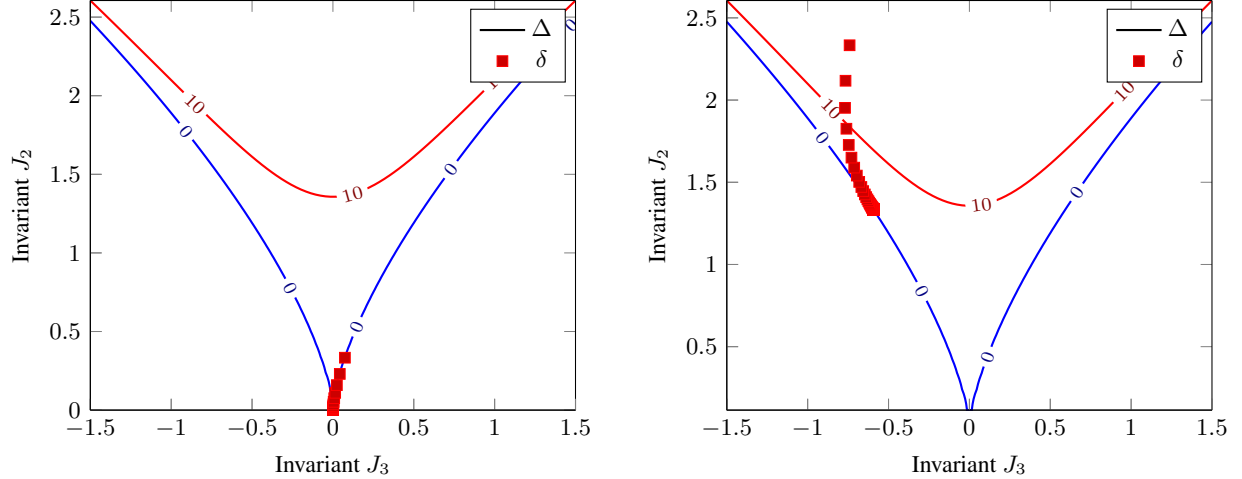
where $\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ is a diagonal matrix with prescribed eigenvalues, and $\mathbf{U}$ is a nonsingular transformation matrix. We evaluate the matrix $\text{fl}(\mathbf{A})$ from Eq. (17) using `numpy.linalg.inv` to compute $\mathbf{U}^{-1}$ and `numpy.matmul` to compute the matrix—matrix products, all in double precision. The resulting matrix $\text{fl}(\mathbf{A})$ is then used as input to the invariant evaluation algorithms. The floating-point matrix $\text{fl}(\mathbf{A})$ is not guaranteed to have the exact eigenvalues $\lambda_1, \lambda_2, \lambda_3$ of the diagonal matrix $\mathbf{D}$. Nevertheless, we compute the forward error of an algorithm f as

$$\text{forward error} = |\text{fl}(f(\text{fl}(\mathbf{A}))) - f_{\text{ref}}(\text{fl}(\mathbf{A}))|. \quad (18)$$

An important detail is that we compute the high-precision reference value $f_{\text{ref}}(\text{fl}(\mathbf{A}))$ at the floating-point matrix $\text{fl}(\mathbf{A})$, not at the exact matrix $\mathbf{A}$.

In order to capture the limit cases of eigenvalue multiplicities, we consider two benchmark paths in this paper, parametrized by a small parameter $\delta \rightarrow 0$. The paths are given by

- $\mathbf{D}_1 = \text{diag}(\lambda_1, \lambda_2, \lambda_3) = \text{diag}(1, 1, 1 + \delta)$, which represents a limiting case of $J_2 \rightarrow 0$ and $J_3 \rightarrow 0$, moving along the double-eigenvalue path towards the triple-eigenvalue,
- $\mathbf{D}_2 = \text{diag}(\lambda_1, \lambda_2, \lambda_3) = \text{diag}(-1, 1, 1 + \delta)$, which represents a limiting case of $\Delta \rightarrow 0$, but both J_3 and J_2 stay finite and away from zero, so we move towards a double-eigenvalue configuration.



(a) Discriminant contour lines with the benchmark path for $\mathbf{D}_1 = \text{diag}(1, 1, 1 + \delta)$. This represents a limiting case of $J_2 \rightarrow 0$ in which each generated matrix has $\Delta = 0$, meaning we move along the double-eigenvalue path towards the triple-eigenvalue.

(b) Discriminant contour lines with the benchmark path for $\mathbf{D}_2 = \text{diag}(-1, 1, 1 + \delta)$. This represents a limiting case of $\Delta \rightarrow 0$, but both J_3 and J_2 stay finite and away from zero, so we move towards a double-eigenvalue configuration.

Figure 2: Benchmark cases in this paper. The red squares represent the limiting path $\delta \rightarrow 0$.

The two benchmark paths are illustrated in the J_3 – J_2 plane in Fig. 2.

Transformation matrices $\mathbf{U}$ used in the benchmarks are chosen as

$$\mathbf{U}_{\text{symm}} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}, \quad \mathbf{U}_1 = \begin{bmatrix} 1 & -1 & 1 \\ 1 & 1 & 1 \\ -1 & -1 & 1 \end{bmatrix}, \quad \mathbf{U}_2(\gamma) = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 1 \\ 2 & 1 & 2 + \gamma \end{bmatrix}. \quad (19)$$

Transformation matrix $\mathbf{U}_{\text{symm}}$ represents an orthogonal transformation, so the 2-norm condition number is $\kappa_2(\mathbf{U}_{\text{symm}}) = 1$ and, as a consequence, any matrix of the form (17) is symmetric.

The matrix $\mathbf{U}_1$ represents a nonorthogonal transformation matrix with small 2-norm condition number $\kappa_2(\mathbf{U}_1) = 2$. Matrices of the form (17) with $\mathbf{U} = \mathbf{U}_1$ are nonsymmetric but have a well-conditioned eigenbasis.

The third case of matrix $\mathbf{U}_2(\gamma)$ represents a nonorthogonal transformation matrix with tunable condition number $\kappa_2(\mathbf{U}_2)$. One can show that $\kappa_2(\mathbf{U}_2(\gamma)) \rightarrow \infty$ as $\gamma \rightarrow 0$ (as the rows become linearly dependent). Matrices of the form (17) with $\mathbf{U} = \mathbf{U}_2(\gamma)$ are nonsymmetric and can have an arbitrarily ill-conditioned eigenbasis. They represent the most challenging case for numerical evaluation of invariants and eigenvalues.

4 Invariant I_1

The first invariant I_1 is defined as the trace of the matrix,

$$I_1(\mathbf{A}) := \text{tr}(\mathbf{A}) = A_{00} + A_{11} + A_{22}. \quad (20)$$

The algorithm for evaluating I_1 sums the diagonal elements, as shown in Algorithm 1.

Algorithm 1 Evaluation of the invariant I_1

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$
 $I_1 = A_{00} + A_{11} + A_{22}$
return I_1

Algorithm 1 is trivially backward stable, as the sum of three floating-point numbers can be seen as the exact sum of slightly perturbed inputs. The floating-point evaluation reads

$$\begin{aligned} \text{fl}(I_1) &= ((A_{00} + A_{11})(1 + \delta_0) + A_{22})(1 + \delta_1) \\ &= A_{00}(1 + \delta_0)(1 + \delta_1) + A_{11}(1 + \delta_0)(1 + \delta_1) + A_{22}(1 + \delta_1) \\ &= A_{00}(1 + \theta_2) + A_{11}(1 + \theta_2) + A_{22}(1 + \theta_1). \end{aligned} \quad (21)$$

This is equivalent to a diagonal perturbation of the input matrix $\mathbf{A}$ with

$$\text{fl}(I_1) = I_1(\mathbf{A} + \delta\mathbf{A}), \quad \text{where} \quad \delta\mathbf{A} = \begin{bmatrix} A_{00}\theta_2 & 0 & 0 \\ 0 & A_{11}\theta_2 & 0 \\ 0 & 0 & A_{22}\theta_1 \end{bmatrix}. \quad (22)$$

The perturbation is componentwise relatively small, i.e.,

$$\frac{|\delta A_{ij}|}{|A_{ij}|} \leq C\epsilon_{\text{mach}}, \quad (23)$$

with $C = 3$ for all $i, j \in \{0, 1, 2\}$.

Since the directional derivative of the trace is

$$\frac{\partial}{\partial \mathbf{A}} \text{tr}(\mathbf{A})[\delta\mathbf{A}] = \text{tr}(\delta\mathbf{A}) = \langle \mathbf{I}, \delta\mathbf{A} \rangle, \quad (24)$$

we have that the Jacobian is $\mathbf{J}_{I_1} = \mathbf{I}$. This implies the following result: Algorithm 1 for evaluating I_1 is forward stable in the sense that the absolute forward error satisfies

$$|\text{fl}(I_1) - I_1| \leq C\|\mathbf{A}\|\epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2), \quad (25)$$

where C is a moderate constant. The relative condition number of I_1 is unbounded, as

$$\kappa_{I_1}(\mathbf{A}) = \|\mathbf{J}_{I_1}\| \frac{\|\mathbf{A}\|}{|I_1|} = \frac{C\|\mathbf{A}\|}{|\text{tr}(\mathbf{A})|} \quad (26)$$

where constant C depends on the chosen matrix norm. Thus, we cannot expect any algorithm to be accurate when $|\text{tr}(\mathbf{A})|$ is small compared to $\|\mathbf{A}\|$.

5 Invariant J_2

Lemma 5.1. *The Jacobian of the J_2 invariant is*

$$\mathbf{J}_{J_2}(\mathbf{A}) = \text{dev}(\mathbf{A})^\top. \quad (27)$$

As a consequence, the J_2 invariant is well-conditioned in the absolute sense for matrices whose deviatoric part has small norm.

Proof. We use the following directional derivative

$$\frac{\partial}{\partial \mathbf{A}} \text{dev}(\mathbf{A})[\delta\mathbf{A}] = \text{dev}(\delta\mathbf{A}), \quad (28)$$

which follows from the linearity of the deviatoric operator. Combining this with the definition of J_2 , we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{A}} J_2[\delta\mathbf{A}] &= \frac{\partial}{\partial \mathbf{A}} \frac{1}{2} \text{tr}(\text{dev}(\mathbf{A})^2)[\delta\mathbf{A}] \\ &= \frac{1}{2} \text{tr}(\text{dev}(\mathbf{A}) \text{dev}(\delta\mathbf{A}) + \text{dev}(\delta\mathbf{A}) \text{dev}(\mathbf{A})) \\ &= \text{tr}(\text{dev}(\mathbf{A}) \text{dev}(\delta\mathbf{A})) \\ &= \langle \text{dev}(\mathbf{A})^\top, \text{dev}(\delta\mathbf{A}) \rangle = \langle \text{dev}(\mathbf{A})^\top, \delta\mathbf{A} \rangle. \end{aligned} \quad (29)$$

□

Algorithm 2 Evaluation of the invariant J_2

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$d_0 = A_{00} - A_{11}, d_1 = A_{00} - A_{22}, d_2 = A_{11} - A_{22}$$

 $\triangleright$ Diagonal differences

$$\text{offdiag} = A_{01}A_{10} + A_{02}A_{20} + A_{12}A_{21}$$

 $\triangleright$ Off-diagonal products

$$\text{diag} = \frac{1}{6}(d_0^2 + d_1^2 + d_2^2)$$

 $\triangleright$ Sum of squares of diagonal differences

$$J_2 = \text{diag} + \text{offdiag}$$

return J_2

Lemma 5.2. *Algorithm 2 for evaluating J_2 is backward stable in the componentwise relative sense.*

Proof. We note that the final expression for J_2 is a sum of two terms, where the first one is based on off-diagonal products and the second one is a sum of squares of the diagonal differences. Let us examine the sum of squares of the diagonal differences first. Diagonal differences are computed as

$$\begin{aligned} \text{fl}(d_0) &= \text{fl}(A_{00} - A_{11}) = (A_{00} - A_{11})(1 + \delta_0), \\ \text{fl}(d_1) &= \text{fl}(A_{00} - A_{22}) = (A_{00} - A_{22})(1 + \delta_1), \\ \text{fl}(d_2) &= \text{fl}(A_{11} - A_{22}) = (A_{11} - A_{22})(1 + \delta_2). \end{aligned} \quad (30)$$

and, using Higham's θ -notation, we have

$$\text{fl}(\text{diag}) = \frac{1}{6} (d_0^2(1 + \theta_6) + d_1^2(1 + \theta'_6) + d_2^2(1 + \theta_5)). \quad (31)$$

The largest relative error here is $(1 + \theta_6)$, since the first diagonal difference d_0 incurs errors from the subtraction itself, squaring, two additions to the other diagonal differences, and one division by 6.

Each off-diagonal product produces a single roundoff error, and summing them together with the diagonal term yields

$$\begin{aligned} \text{fl}(J_2) &= A_{01}A_{10}(1 + \theta_5) + A_{02}A_{20}(1 + \theta'_5) + A_{12}A_{21}(1 + \theta_4) \\ &\quad + \frac{1}{6} (d_0^2(1 + \theta_7) + d_1^2(1 + \theta'_7) + d_2^2(1 + \theta_6)). \end{aligned} \quad (32)$$

Here, we already recognize the perturbations required for the off-diagonal terms, i.e.,

$$\delta \mathbf{A} = \begin{bmatrix} A_{00}\alpha & A_{01}\theta_5 & A_{02}\theta'_5 \\ 0 & A_{11}\alpha & A_{12}\theta_4 \\ 0 & 0 & A_{22}\alpha \end{bmatrix} \quad (33)$$

while α for the diagonal perturbation is to be determined. For the exact computation with the perturbed input, we have

$$\begin{aligned} J_2(\mathbf{A} + \delta \mathbf{A}) &= A_{01}A_{10}(1 + \theta_5) + A_{02}A_{20}(1 + \theta'_5) + A_{12}A_{21}(1 + \theta_4) \\ &\quad + \frac{1}{6} (1 + \alpha)^2 (d_0^2 + d_1^2 + d_2^2). \end{aligned} \quad (34)$$

To match the diagonal contributions we need α such that

$$(1 + \alpha)^2 (d_0^2 + d_1^2 + d_2^2) = d_0^2(1 + \theta_7) + d_1^2(1 + \theta'_7) + d_2^2(1 + \theta_6) \quad (35)$$

which is satisfied for

$$\alpha = \sqrt{\frac{d_0^2(1 + \theta_7) + d_1^2(1 + \theta'_7) + d_2^2(1 + \theta_6)}{d_0^2 + d_1^2 + d_2^2}} - 1. \quad (36)$$

A bound on α follows from the first-order Taylor expansion $\sqrt{1 + x} = 1 + x/2 + \mathcal{O}(x^2)$ for $x \approx 0$:

$$\begin{aligned} |\alpha| &= \left| \sqrt{1 + \frac{d_0^2\theta_7 + d_1^2\theta'_7 + d_2^2\theta_6}{d_0^2 + d_1^2 + d_2^2}} - 1 \right| \\ &= \left| 1 + \frac{1}{2}\xi + \mathcal{O}(\xi^2) - 1 \right| \quad (\text{Taylor expansion}) \\ &= \frac{1}{2}|\xi| + |\mathcal{O}(\xi^2)| \leq \frac{1}{2}\gamma_7 + |\mathcal{O}(\gamma_7^2)|. \quad (\text{see below}) \end{aligned} \quad (37)$$

The last inequality follows from

$$|\xi| = \left| \frac{d_0^2 \theta_7 + d_1^2 \theta_7' + d_2^2 \theta_6}{d_0^2 + d_1^2 + d_2^2} \right| \leq \left| \frac{\max(\theta_7, \theta_7', \theta_6)(d_0^2 + d_1^2 + d_2^2)}{d_0^2 + d_1^2 + d_2^2} \right| \leq \gamma_7 \quad (38)$$

since the squared diagonal differences are nonnegative.

From the way we constructed the perturbation (i.e., relative to the matrix entries) we now have the componentwise relative backward error result

$$\text{fl}(J_2(\mathbf{A})) = J_2(\mathbf{A} + \delta \mathbf{A}), \quad \text{where} \quad \frac{|\delta \mathbf{A}_{ij}|}{|\mathbf{A}_{ij}|} \leq C \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2), \quad (39)$$

for all i, j , with $C \approx 5$. □

Theorem 5.3. *Algorithm 2 is forward stable in the sense that the absolute forward error satisfies*

$$|\text{fl}(J_2) - J_2| \leq C \|\text{dev}(\mathbf{A})\|^2 \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (40)$$

Proof. This is a consequence of the Jacobian and the backward stability result. Combining Lemmas 5.1 and 5.2, we have

$$\begin{aligned} |\text{fl}(J_2) - J_2| &= |J_2(\mathbf{A} + \delta \mathbf{A}) - J_2(\mathbf{A})| && \text{(Lemma 5.2)} \\ &= \left| \langle \text{dev}(\mathbf{A})^\top, \delta \mathbf{A} \rangle + \mathcal{O}(\|\delta \mathbf{A}\|^2) \right| && \text{(Taylor expansion, Lemma 5.1)} \\ &= \left| \langle \text{dev}(\mathbf{A})^\top, \text{dev}(\delta \mathbf{A}) \rangle + \mathcal{O}(\|\delta \mathbf{A}\|^2) \right| && (41) \\ &\leq \|\text{dev}(\mathbf{A})^\top\|_2 \|\text{dev}(\delta \mathbf{A})\|_2 + \mathcal{O}(\|\delta \mathbf{A}\|^2). && \text{(Cauchy-Schwarz)} \end{aligned}$$

At this point, we need to show that the componentwise relative backward error bound from Eq. (39) implies a normwise bound on the deviatoric part. This is not true in general, but we use the specific structure of the perturbation $\delta \mathbf{A}$ from Eq. (33). First, we notice that $\|\text{diag}(\text{dev}(\delta \mathbf{A}))\| = |\alpha| \|\text{diag}(\text{dev}(\mathbf{A}))\|$, where $\text{diag}(\cdot)$ denotes the diagonal part of a matrix. In addition, for any matrix,

$$\|\mathbf{B}\|_F^2 = \|\text{diag}(\mathbf{B})\|_F^2 + \|\text{offdiag}(\mathbf{B})\|_F^2, \quad (42)$$

since the diagonal and off-diagonal parts are orthogonal in the Frobenius inner product. We can write

$$\begin{aligned} \|\text{dev}(\delta \mathbf{A})\|_F^2 &= \|\text{diag}(\text{dev}(\delta \mathbf{A}))\|_F^2 + \|\text{offdiag}(\text{dev}(\delta \mathbf{A}))\|_F^2 \\ &\leq \alpha^2 \|\text{diag}(\text{dev}(\mathbf{A}))\|_F^2 + \max(\theta_5, \theta_5')^2 \|\text{offdiag}(\text{dev}(\mathbf{A}))\|_F^2 \\ &\leq \max(\alpha, \theta_5, \theta_5')^2 \left(\|\text{diag}(\text{dev}(\mathbf{A}))\|_F^2 + \|\text{offdiag}(\text{dev}(\mathbf{A}))\|_F^2 \right) \\ &= \max(\alpha, \theta_5, \theta_5')^2 \|\text{dev}(\mathbf{A})\|_F^2. \end{aligned} \quad (43)$$

By norm equivalence in finite-dimensional spaces we obtain

$$\|\text{dev}(\delta \mathbf{A})\| \leq C \|\text{dev}(\mathbf{A})\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (44)$$

Plugging this into Eq. (41) gives

$$|\text{fl}(J_2) - J_2| \leq C \|\text{dev}(\mathbf{A})\|^2 \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (45)$$

□

Lemma 5.4. *Let $\mathbf{A} = \mathbf{U} \mathbf{D} \mathbf{U}^{-1}$ be a real, diagonalizable 3×3 matrix with real spectrum. Then*

$$\frac{2}{9\kappa_2^2} J_2 \leq \|\text{dev}(\mathbf{A})\|_F^2 \leq 18\kappa_2^2 J_2, \quad (46)$$

where $\kappa_2 = \|\mathbf{U}\|_2 \|\mathbf{U}^{-1}\|_2$ is the spectral condition number of the matrix $\mathbf{U}$.

Proof. Let $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1}$ with $\mathbf{D} = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$ and $\lambda_i \in \mathbb{R}$. Denote the mean eigenvalue by $\bar{\lambda} = \frac{1}{3} \sum_{i=1}^3 \lambda_i = \frac{1}{3} \text{tr}(\mathbf{A})$ and define the centered eigenvalues $\mu_i = \lambda_i - \bar{\lambda}$ (so $\sum_i \mu_i = 0$). The deviatoric part of $\mathbf{A}$ is

$$\mathbf{S} := \text{dev}(\mathbf{A}) = \mathbf{A} - \bar{\lambda}\mathbf{I} = \mathbf{U}(\mathbf{D} - \bar{\lambda}\mathbf{I})\mathbf{U}^{-1} = \mathbf{U} \text{diag}(\mu_1, \mu_2, \mu_3) \mathbf{U}^{-1}. \quad (47)$$

Using similarity invariance of the trace, we have

$$2J_2 = \text{tr}(\mathbf{S}^2) = \text{tr}(\mathbf{U} \text{diag}(\mu)^2 \mathbf{U}^{-1}) = \text{tr}(\text{diag}(\mu)^2) = \sum_{i=1}^3 \mu_i^2 = \|\text{diag}(\mu)\|_F^2 \quad (48)$$

where $\text{diag}(\mu) := \text{diag}(\mu_1, \mu_2, \mu_3)$. Hence

$$\|\text{diag}(\mu)\|_F = \sqrt{2J_2}. \quad (49)$$

For any matrices $\mathbf{A}, \mathbf{X}, \mathbf{B}$, the inequality

$$\|\mathbf{A}\mathbf{X}\mathbf{B}\|_F \leq \|\mathbf{A}\|_F \|\mathbf{X}\|_F \|\mathbf{B}\|_F \quad (50)$$

holds, since the Frobenius norm is submultiplicative. Applying this with $\mathbf{A} = \mathbf{U}$, $\mathbf{X} = \text{diag}(\mu)$, and $\mathbf{B} = \mathbf{U}^{-1}$,

$$\|\mathbf{S}\|_F = \|\mathbf{U} \text{diag}(\mu) \mathbf{U}^{-1}\|_F \leq \|\mathbf{U}\|_F \|\text{diag}(\mu)\|_F \|\mathbf{U}^{-1}\|_F = 3\kappa_2 \sqrt{2J_2}. \quad (51)$$

We used the norm equivalence and the upper bound $\|\mathbf{U}\|_F \leq \sqrt{3}\|\mathbf{U}\|_2$ for any 3×3 matrix and the spectral norm $\|\cdot\|_2$. Squaring gives the upper bound $\|\text{dev}(\mathbf{A})\|_F^2 \leq 18\kappa_2^2 J_2$.

For the lower bound, rewrite $\text{diag}(\mu) = \mathbf{U}^{-1}\mathbf{S}\mathbf{U}$ and apply the same inequality

$$\|\text{diag}(\mu)\|_F = \|\mathbf{U}^{-1}\mathbf{S}\mathbf{U}\|_F \leq \|\mathbf{U}^{-1}\|_F \|\mathbf{S}\|_F \|\mathbf{U}\|_F = 3\kappa_2 \|\mathbf{S}\|_F. \quad (52)$$

Hence

$$\|\mathbf{S}\|_F \geq \frac{\|\text{diag}(\mu)\|_F}{3\kappa_2} = \frac{\sqrt{2J_2}}{3\kappa_2} \quad (53)$$

and squaring yields

$$\|\text{dev}(\mathbf{A})\|_F^2 \geq \frac{2}{9\kappa_2^2} J_2. \quad (54)$$

□

Corollary 5.5. For a real, diagonalizable 3×3 matrix $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1}$ with real spectrum, Algorithm 2 satisfies

$$|\text{fl}(J_2) - J_2| \leq C \kappa_2^2 J_2 \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2), \quad (55)$$

where $\kappa_2 = \|\mathbf{U}\|_2 \|\mathbf{U}^{-1}\|_2$ is the spectral condition number of $\mathbf{U}$. In particular, if $\mathbf{A}$ is symmetric (such that $\kappa_2 = 1$), the algorithm is accurate.

Proof. This is a consequence of Lemma 5.4 and Theorem 5.3. □

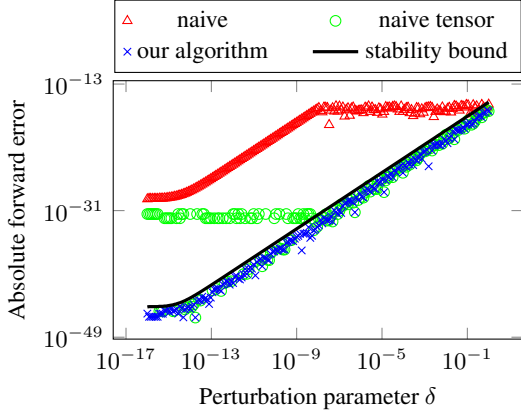
Remark (nonnormality and Henrici). For nonnormal matrices, Henrici's departure from normality considers the Schur form $\mathbf{A} = \mathbf{Q}(\mathbf{D} + \mathbf{N})\mathbf{Q}^T$ with $\mathbf{Q}$ orthogonal, $\mathbf{D}$ (block-)diagonal, and $\mathbf{N}$ strictly upper triangular, and defines the nonnormality measure as $\nu(\mathbf{A}) := \|\mathbf{N}\|_F$, so that $\nu(\mathbf{A}) = 0$ iff $\mathbf{A}$ is normal. In practice, large $\nu(\mathbf{A})$ is often accompanied by ill-conditioned eigenvectors (large $\kappa = \|\mathbf{U}\| \|\mathbf{U}^{-1}\|$), which explains the κ^2 amplification appearing in our bounds.¹

Remark (relative deviatoric conditioning). The relative condition number as defined in Eq. (12) is not informative for the invariant J_2 . For $\mathbf{A} = \text{diag}(1, 1, 1 + \delta)$, as $\delta \rightarrow 0$ we have

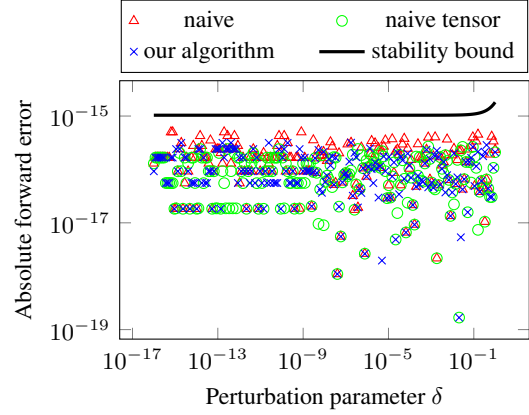
$$\kappa_{J_2}(\mathbf{A}) = \|\mathbf{J}_{J_2}\| \frac{\|\mathbf{A}\|}{|J_2|} = \frac{\|\text{dev}(\mathbf{A})\|_F}{\frac{1}{2}\|\text{dev}(\mathbf{A})\|_F^2} \|\mathbf{A}\|_F = \frac{2\|\mathbf{A}\|_F}{\|\text{dev}(\mathbf{A})\|_F} \rightarrow \infty, \quad (56)$$

where we used Lemma 5.1 and, for symmetric $\mathbf{A}$, $2J_2 = \|\text{dev}(\mathbf{A})\|_F^2$. Nevertheless, Corollary 5.5 shows that the algorithm is accurate for this symmetric case. Inspecting the proof of Theorem 5.3 reveals that it is the deviatoric part

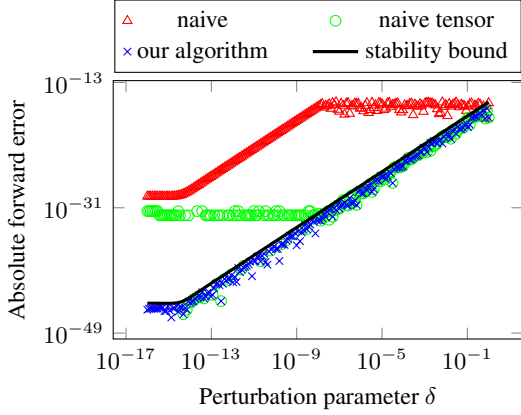
¹<https://nhigam.com/2020/11/24/what-is-a-nonnormal-matrix/>



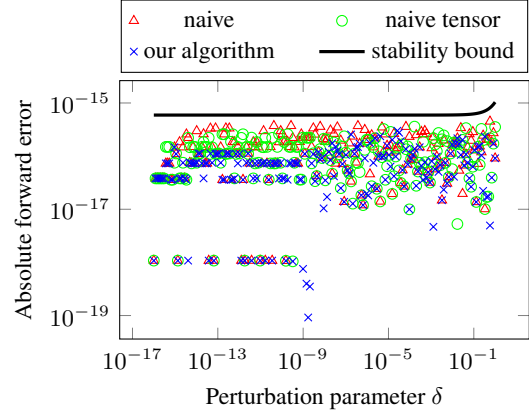
(a) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



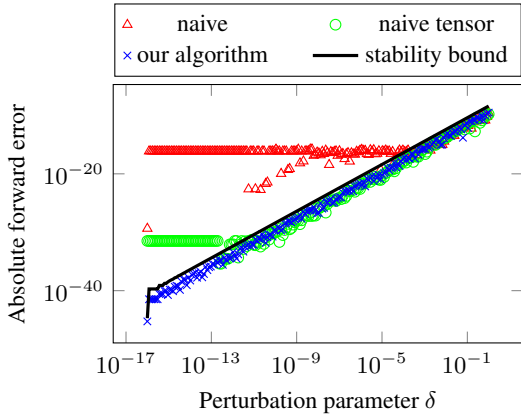
(b) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



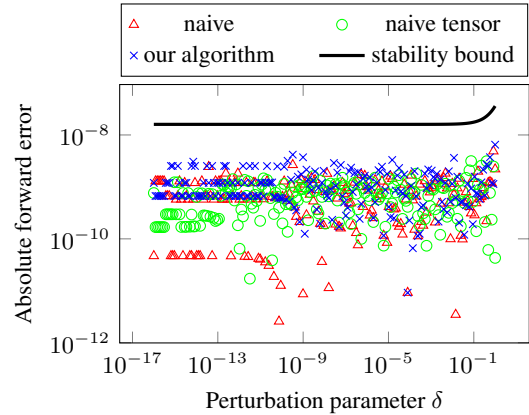
(c) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(d) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(e) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).



(f) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).

Figure 3: Numerical stability analysis for the invariant J_2 .

of the perturbation that must be controlled, rather than the full perturbation $\delta \mathbf{A}$. Motivated by this we define the *relative deviatoric condition number*

$$\kappa_f^{\text{dev}}(\mathbf{A}) := \|\mathbf{J}_f\| \frac{\|\text{dev}(\mathbf{A})\|}{\|f(\mathbf{A})\|}. \quad (57)$$

For $f = J_2$ and symmetric $\mathbf{A}$,

$$\kappa_{J_2}^{\text{dev}}(\mathbf{A}) = \frac{\|\text{dev}(\mathbf{A})\|_F}{\frac{1}{2}\|\text{dev}(\mathbf{A})\|_F^2} \|\text{dev}(\mathbf{A})\|_F = 2, \quad (\text{for symmetric } \mathbf{A}), \quad (58)$$

so J_2 is well-conditioned in the relative deviatoric sense for symmetric matrices.

The backward stability notion used to derive the forward error bound also needs refinement. What is required in the proof of Theorem 5.3 (in the first-order term) is

$$\frac{\|\text{dev}(\delta \mathbf{A})\|}{\|\text{dev}(\mathbf{A})\|} \leq C \epsilon_{\text{mach}}, \quad (59)$$

which we term *relative deviatoric backward stability*. This notion is neither stronger nor weaker than componentwise relative backward stability. We demonstrate this by two examples.

First, a perturbation that is componentwise relative stable but not relative deviatoric stable

$$\delta \mathbf{A} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \epsilon_{\text{mach}} \end{bmatrix} \quad \text{for} \quad \mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (60)$$

has clearly each component small relative to $\mathbf{A}$, but $\|\text{dev}(\delta \mathbf{A})\|$ is of order ϵ_{mach} while $\|\text{dev}(\mathbf{A})\| = 0$.

Second, a perturbation that is relative deviatoric stable but not componentwise stable

$$\delta \mathbf{A} = \mathbf{I} \quad \text{for} \quad \mathbf{A} = \mathbf{I} \quad (61)$$

has $\|\text{dev}(\delta \mathbf{A})\| = 0$ so the relative deviatoric condition is satisfied, but the componentwise relative error is of order 1.

Corollary 5.6. *Algorithm 2 is backward stable in the relative deviatoric sense.*

Proof. This is a consequence of the proof of Lemma 5.2 and the discussion in the proof of Theorem 5.3. $\square$

Remark (intuition behind Algorithm 2). Algorithm 2 evaluates J_2 by first forming the diagonal differences. This step is crucial for numerical stability near $J_2 = 0$. In particular, it guarantees

$$\text{fl}(J_2(\alpha \mathbf{I})) = 0, \quad (62)$$

so the algorithm is exact for the scaled identity matrix. In fact, for this to hold we need to show that the quadratic term in Corollary 5.5 vanishes as well. This is a consequence of the second directional derivative of J_2 (i.e., action of the Hessian) being

$$\frac{\partial^2}{\partial \mathbf{A}^2} J_2[\delta \mathbf{A}, \delta \mathbf{A}] = \langle \text{dev}(\delta \mathbf{A})^\top, \text{dev}(\delta \mathbf{A}) \rangle, \quad (63)$$

but the relative deviatoric backward stability then implies that

$$|\langle \text{dev}(\delta \mathbf{A})^\top, \text{dev}(\delta \mathbf{A}) \rangle| \leq C \|\text{dev}(\mathbf{A})\|^2 \epsilon_{\text{mach}}^2 \quad (64)$$

so the quadratic term vanishes for $\mathbf{A} = \alpha \mathbf{I}$.

As seen in Theorem 5.3, this behavior is a necessary consequence of any algorithm that is backward stable in the relative deviatoric sense. Moreover, in the vicinity of the scaled identity, for example for some $\mathbf{A} = \alpha \mathbf{I} + \mathbf{E}$ where $\mathbf{E}$ is elementwise of order ϵ_{mach} , the diagonal differences and the off-diagonal products are all of order ϵ_{mach} . This prevents catastrophic cancellation and leads to the expression for J_2 that is of order ϵ_{mach}^2 .

5.1 Discussion of the numerical benchmarks

There are three different implementations of evaluation of J_2 benchmarked in Fig. 3: *naive*, *naive tensor*, and *our algorithm*.

Naive approach is based on Algorithm 3, which is an unrolled polynomial expression (monomial sum). There is no structure-exploiting rearrangement of terms, so this algorithm is expected to be numerically unstable. The second

implementation is called *naive tensor* and is based on the definition of J_2 as $J_2 = \frac{1}{2} \text{tr}(\text{dev}(\mathbf{A})^2)$, where all operations are computed via a tensor implementation in NumPy. The algorithm is listed in Algorithm 4, where the trace is computed using `numpy.trace` (NumPy, 2025d) and matrix multiplication using `numpy.matmul` (NumPy, 2025c).

Algorithm 3 Naive evaluation of the invariant J_2

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$J_2 = \frac{1}{3}(A_{00}^2 - A_{00}A_{11} - A_{00}A_{22} + 3A_{01}A_{10} + 3A_{02}A_{20} + A_{11}^2 - A_{11}A_{22} + 3A_{12}A_{21} + A_{22}^2)$$

return J_2

Algorithm 4 Naive tensor evaluation of the invariant J_2

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$\mathbf{S} = \mathbf{A} - \frac{1}{3} \text{tr}(\mathbf{A})\mathbf{I}$$

▷ Deviatoric part

$$J_2 = \frac{1}{2} \text{tr}(\mathbf{S}^2)$$

▷ Matrix multiplication and trace

return J_2

The *naive* implementation shows the largest forward errors in all benchmark cases, as seen in Fig. 3.

The *naive tensor* implementation is more accurate than the naive one, but still shows large forward errors, especially in Figs. 3a and 3c. The reason is that the deviatoric part is computed based on the trace shift. Computation of the trace introduces rounding errors, which then prevent the deviatoric part from being exactly zero even for the scaled identity matrix.

Lastly, results for *our algorithm* (Algorithm 2) are included. This algorithm shows the best accuracy in all benchmarks. In all cases the stability bound from Theorem 5.3 is satisfied. This is true even for the most challenging case of the transformation matrix being nonorthogonal and nearly singular, $\mathbf{U} = \mathbf{U}_2(\gamma)$. The γ parameter was chosen as $\gamma = 10^{-3}$, which leads to condition number $\kappa_2(\mathbf{U}_2) \approx 9 \times 10^3$. The benchmark case of Fig. 2a has J_2 approaching zero, so in order to achieve the accuracy promised by Corollary 5.5, the absolute forward error must decrease proportionally. This is observed in Figs. 3a, 3e and 3c. For the case of $\mathbf{D} = \text{diag}(1, 1, 1 + \delta)$ and $\delta \approx 10^{-16}$ the exact value of $J_2 \approx \delta^2 = 10^{-32}$. The accurate algorithm must compute this value with relative error of order $\epsilon_{\text{mach}} \approx 10^{-16}$, which means an absolute error of order 10^{-48} . This is indeed achieved in Fig. 3c and for the well-conditioned case of Fig. 3a.

Note, that the included stability bound plots (solid lines) in Fig. 3 are based only on the lowest order term from Eq. (45), i.e., $\|\text{dev}(\mathbf{A})\|_F^2 \epsilon_{\text{mach}}$. Following the discussion in the remark about relative deviatoric conditioning, the higher order terms are proportional to $\|\text{dev}(\mathbf{A})\|_F^2 \epsilon_{\text{mach}}^2$ so they are negligible.

6 Invariant J_3

Lemma 6.1. *The Jacobian of the J_3 invariant is given by*

$$\mathbf{J}_{J_3} = \text{dev}(\text{cof}(\text{dev}(\mathbf{A}))). \quad (65)$$

Proof. Using

$$\frac{\partial}{\partial \mathbf{A}} \det(\mathbf{A})[\delta \mathbf{A}] = \langle \text{cof}(\mathbf{A}), \delta \mathbf{A} \rangle \quad (66)$$

and the linearity of the deviatoric operator Eq. (28), we have

$$\begin{aligned} \frac{\partial}{\partial \mathbf{A}} J_3[\delta \mathbf{A}] &= \frac{\partial}{\partial \mathbf{A}} \det(\text{dev}(\mathbf{A}))[\delta \mathbf{A}] \\ &= \langle \text{cof}(\text{dev}(\mathbf{A})), \text{dev}(\delta \mathbf{A}) \rangle \\ &= \langle \text{dev}(\text{cof}(\text{dev}(\mathbf{A}))), \delta \mathbf{A} \rangle. \end{aligned} \quad (67)$$

□

Motivated by the observations in Section 5, we propose the following algorithm for evaluating J_3 .

Algorithm 5 Evaluation of the J_3 invariant

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$d_0 = A_{00} - A_{11}, d_1 = A_{00} - A_{22}, d_2 = A_{11} - A_{22}$ $t_1 = d_1 + d_2$ $t_2 = d_0 - d_2$ $t_3 = -d_0 - d_1$ $\text{offdiag} = A_{01}A_{12}A_{20} + A_{02}A_{10}A_{21}$ $\text{mixed} = \frac{1}{3}(A_{01}A_{10}t_1 + A_{02}A_{20}t_2 + A_{12}A_{21}t_3)$ $\text{diag} = \frac{1}{27}t_1t_2t_3$ $J_3 = \text{offdiag} + \text{mixed} - \text{diag}$ return J_3	▷ Diagonal differences ▷ Off-diagonal products ▷ Mixed products ▷ Product of diagonal differences
---	---

Algorithm 5 is an expansion of the determinant of the deviatoric part of $\mathbf{A}$ expressed in terms of diagonal differences, off-diagonal products, and mixed products. Similar to the J_2 invariant, J_3 approaches zero as the matrix $\mathbf{A}$ approaches a scaled identity matrix. This is the reason for forming the diagonal differences.

However, the J_3 invariant approaches zero in a more general case, when the deviatoric part of $\mathbf{A}$ becomes singular. Consider an example matrix

$$\mathbf{A} = \text{diag}(1, 2, 3) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{bmatrix}. \quad (68)$$

This matrix is symmetric, but its deviatoric part is singular, i.e., $J_3 = \det(\text{dev}(\mathbf{A})) = 0$. The diagonal differences in floating-point arithmetic are computed as

$$\text{fl}(d_0) = \text{fl}(1 - 2) = -1(1 + \delta_0), \quad \text{fl}(d_1) = \text{fl}(1 - 3) = -2(1 + \delta_1), \quad \text{fl}(d_2) = \text{fl}(2 - 3) = -1(1 + \delta_2), \quad (69)$$

where $|\delta_i| \leq \epsilon_{\text{mach}}$. The diagonal combination t_2 is computed as

$$\text{fl}(t_2) = \text{fl}(d_0 - d_2) = \text{fl}(-1(1 + \delta_0) + 1(1 + \delta_2)) = (\delta_2 - \delta_0)(1 + \delta_3), \quad |\delta_3| \leq \epsilon_{\text{mach}}. \quad (70)$$

This shows that the relative forward error $|\text{fl}(t_2) - t_2|/|t_2|$ is unbounded. As a consequence, Algorithm 5 does not produce an exact zero for exactly singular deviatoric matrices and cannot be considered accurate in those cases.

Assume we have an algorithm for $\text{fl}(J_3)$ that is backward stable in the relative deviatoric sense, i.e.,

$$\text{fl}(J_3(\mathbf{A})) = J_3(\mathbf{A} + \delta\mathbf{A}), \quad \text{with} \quad \frac{\|\text{dev}(\delta\mathbf{A})\|}{\|\text{dev}(\mathbf{A})\|} \leq C\epsilon_{\text{mach}}, \quad (71)$$

for some constant C . Then we proceed similarly to the proof of Lemma 5.2 and combine the backward error with the Jacobian

$$\begin{aligned} |\text{fl}(J_3) - J_3| &= |J_3(\mathbf{A} + \delta\mathbf{A}) - J_3(\mathbf{A})| \\ &= \left| \langle \text{dev}(\text{cof}(\text{dev}(\mathbf{A}))), \delta\mathbf{A} \rangle + \mathcal{O}(\|\delta\mathbf{A}\|^2) \right| \\ &\leq \|\text{dev}(\text{cof}(\text{dev}(\mathbf{A})))\| \|\text{dev}(\delta\mathbf{A})\| + \mathcal{O}(\|\delta\mathbf{A}\|^2) \\ &\leq C \|\text{dev}(\text{cof}(\text{dev}(\mathbf{A})))\| \|\text{dev}(\mathbf{A})\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \end{aligned} \quad (72)$$

Lemma 6.2. Any deviatoric backward stable algorithm for evaluating J_3 must be forward stable in the sense that the absolute forward error must satisfy

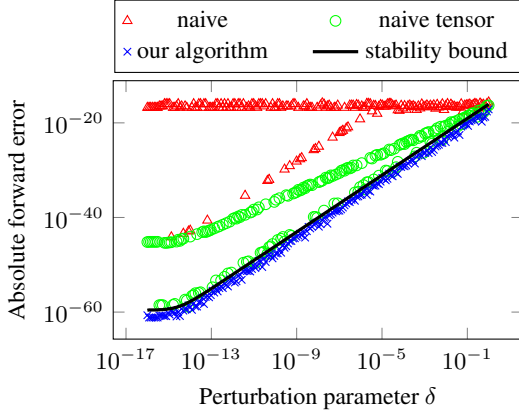
$$|\text{fl}(J_3) - J_3| \leq C \|\text{dev}(\text{cof}(\text{dev}(\mathbf{A})))\| \|\text{dev}(\mathbf{A})\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (73)$$

Proof. Follows directly from Eq. (72). □

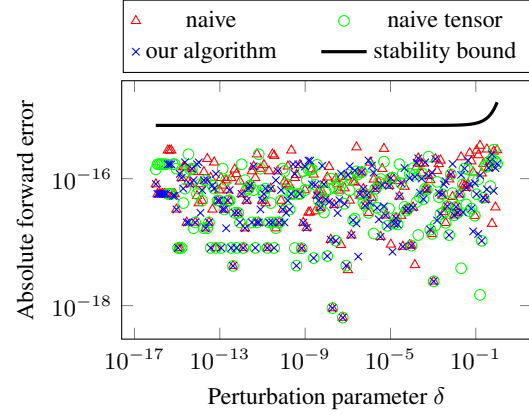
6.1 Discussion of numerical benchmarks

Three different implementations of J_3 are benchmarked in Fig. 4: *naive*, *naive tensor*, and *our algorithm*.

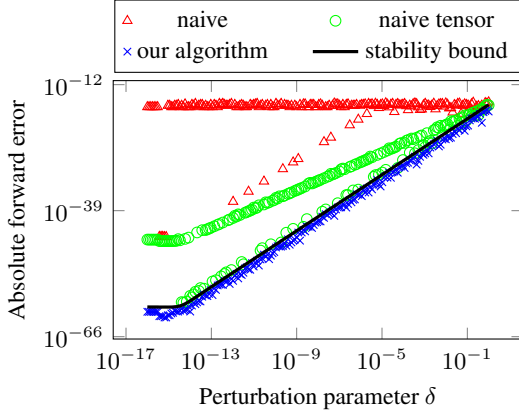
Naive uses Algorithm 6, an unrolled polynomial expression (monomial sum) with no structure-exploiting rearrangement of terms, so it is expected to be numerically unstable. The second implementation, *naive tensor*, is based on the definition $J_3 = \det(\text{dev}(\mathbf{A}))$, where all operations are computed via a tensor implementation in NumPy. The algorithm



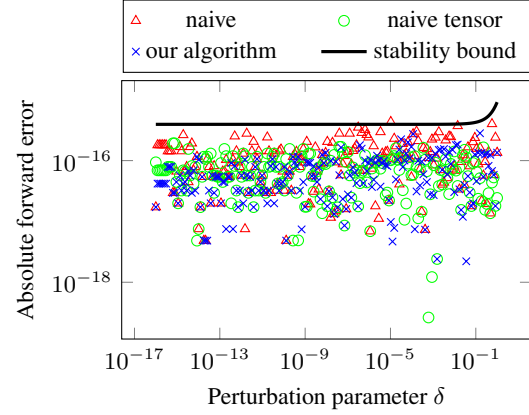
(a) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



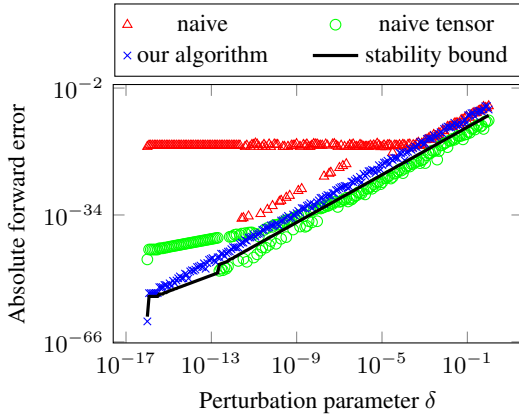
(b) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



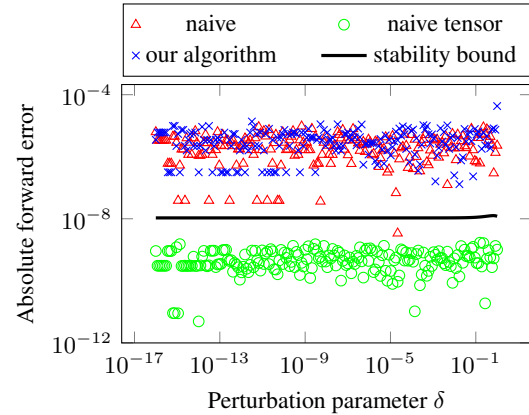
(c) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(d) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(e) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).



(f) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).

Figure 4: Numerical stability analysis for the invariant J_3 .

is listed in Algorithm 7, where the trace is computed using `numpy.trace` (NumPy, 2025d) and matrix multiplication using `numpy.matmul` (NumPy, 2025c).

Algorithm 6 Naive evaluation of the invariant J_3

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$J_3 = \frac{1}{27}(2A_{00}^3 - 3A_{00}^2A_{11} - 3A_{00}^2A_{22} + 9A_{00}A_{01}A_{10} \\ + 9A_{00}A_{02}A_{20} - 3A_{00}A_{11}^2 + 12A_{00}A_{11}A_{22} - 18A_{00}A_{12}A_{21} \\ - 3A_{00}A_{22}^2 + 9A_{01}A_{10}A_{11} - 18A_{01}A_{10}A_{22} + 27A_{01}A_{12}A_{20} \\ + 27A_{02}A_{10}A_{21} - 18A_{02}A_{11}A_{20} + 9A_{02}A_{20}A_{22} + 2A_{11}^3 \\ - 3A_{11}^2A_{22} + 9A_{11}A_{12}A_{21} - 3A_{11}A_{22}^2 + 9A_{12}A_{21}A_{22} + 2A_{22}^3)$$

return J_3

Algorithm 7 Naive tensor evaluation of the invariant J_3

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$\mathbf{S} = \mathbf{A} - \frac{1}{3}\text{tr}(\mathbf{A})\mathbf{I}$$

$$J_3 = \det(\mathbf{S})$$

return J_3

The *naive* implementation shows the largest forward errors in all benchmark cases, as seen in Fig. 4. The error is so large that for $\delta \approx 10^{-16}$ and the well-conditioned case in Fig. 4a the computed J_3 keeps an error of order 10^{-16} , which is 48 orders of magnitude larger than what a forward stable algorithm should produce.

The *naive tensor* implementation is more accurate than the *naive* one, but still shows large forward errors, especially in Figs. 4a and 4c, for the same reasons as explained above for the J_2 invariant. It achieves the best accuracy for badly conditioned cases in Figs. 4e and 4f, probably due to the implementation of the determinant in NumPy based on the numerically stable LU factorization DGETRF from LAPACK (NumPy, 2025a; Anderson et al., 1999).

For *our algorithm* (Algorithm 5), the well-conditioned cases with $\mathbf{U} = \mathbf{U}_1$ (Figs. 4a and 4b) and orthogonal cases with $\mathbf{U} = \mathbf{U}_{\text{symm}}$ (Figs. 4c and 4d) show that the stability bound from Lem. 6.2 is satisfied. This is not true for the most challenging case of the transformation matrix being nonorthogonal and nearly singular, $\mathbf{U} = \mathbf{U}_2(\gamma)$. The γ parameter was chosen as $\gamma = 10^{-3}$, which leads to condition number $\kappa_2(\mathbf{U}_2) \approx 9 \times 10^3$. In this case, the absolute forward error exceeds the stability bound in Figs. 4e and 4f. This suggests that Algorithm 5 is not backward stable in the relative deviatoric sense for all matrices $\mathbf{A}$, but only conditionally stable, depending on the condition number of the eigenbasis.

Note that the included stability bound plots (solid lines) in Fig. 4 are based only on the lowest order term from Eq. (73), i.e., $\|\text{dev}(\text{cof}(\text{dev}(\mathbf{A})))\|_F \|\text{dev}(\mathbf{A})\|_F \epsilon_{\text{mach}}$. It could be shown that the higher order terms are negligible compared to the lowest order term in all benchmark cases.

7 Discriminant Δ

Lemma 7.1. *The Jacobian of the discriminant Δ is given by*

$$\mathbf{J}_\Delta = \text{dev}(12J_2^2\mathbf{A}^\top - 54J_3\text{cof}(\text{dev}(\mathbf{A}))). \quad (74)$$

Proof. Follows from the definition of the discriminant and the Jacobians of the invariants J_2 , Eq. (27), and J_3 , Eq. (65). The deviatoric operator is then moved outside by linearity. $\square$

Motivated by the observations in Section 5 and our previous work, we propose the following algorithm for evaluating Δ . We briefly recall the main ideas from (Habera and Zilian, 2021). We rely on an expression for the discriminant Δ as the determinant of a matrix $\mathbf{B} \in \mathbb{R}^{3 \times 3}$ whose entries are invariants of powers of the matrix $\mathbf{A}$, see (Parlett, 2002). Specifically, we have

$$\Delta = \det(\mathbf{B}), \quad \text{where} \quad B_{ij} = \langle \mathbf{A}^{i-1}, (\mathbf{A}^{j-1})^\top \rangle \quad (75)$$

for $i, j = 1, 2, 3$. In addition, the matrix $\mathbf{B} = \mathbf{X}\mathbf{Y}$ exhibits a factorization into two matrices $\mathbf{X} \in \mathbb{R}^{3 \times 9}$ and $\mathbf{Y} \in \mathbb{R}^{9 \times 3}$, which are built from column- and row-stacked powers of $\mathbf{A}$, respectively. Using the Cauchy–Binet formula, the determinant $\det(\mathbf{X}\mathbf{Y})$ can be expressed as a sum of 14 condensed terms, which we term the *sum-of-products* formula,

$$\Delta = \sum_{i=1}^{14} w_i u_i v_i, \quad (76)$$

where $\mathbf{u} = (u_1, \dots, u_{14})$ and $\mathbf{v} = (v_1, \dots, v_{14})$ are auxiliary vectors built from products of entries of $\mathbf{A}$ and its transpose, respectively, and $\mathbf{w} = (w_1, \dots, w_{14})$ is a vector of integer weights. The important property of the sum-of-products formula is that both factors $\mathbf{u}$ and $\mathbf{v}$ approach zero as the matrix $\mathbf{A}$ approaches a matrix with multiple eigenvalues. This is related to the Cayley–Hamilton theorem and the fact that the columns of $\mathbf{X}$ and rows of $\mathbf{Y}$ become linearly dependent in such situations, since $\mathbf{A}$ satisfies its own minimal polynomial.

Algorithm 8 Evaluation of the discriminant invariant Δ

Require: $\mathbf{A} \in \mathbb{R}^{3 \times 3}$

$$d_0 = A_{00} - A_{11}, d_1 = A_{00} - A_{22}, d_2 = A_{11} - A_{22}$$

$$\mathbf{u} = \text{DX}(\mathbf{A}, d_0, d_1, d_2)$$

$$\mathbf{v} = \text{DX}(\mathbf{A}^\top, d_0, d_1, d_2)$$

$$\mathbf{w} = (9, 6, 6, 6, 8, 8, 8, 2, 2, 2, 2, 2, 2, 1)$$

$$\Delta = \sum_{i=1}^{14} w_i u_i v_i$$

return Δ

function $\text{DX}(\mathbf{M}, d_0, d_1, d_2)$

$$r_1 = M_{01}M_{12}M_{20} - M_{02}M_{10}M_{21}$$

$$r_2 = -M_{01}M_{02}d_2 + M_{01}^2M_{12} - M_{02}^2M_{21}$$

$$r_3 = M_{01}M_{21}d_1 - M_{01}^2M_{20} + M_{02}^2M_{21}^2$$

$$r_4 = M_{02}M_{12}d_0 + M_{01}M_{12}^2 - M_{02}^2M_{10}$$

$$r_5 = M_{01}M_{12}d_1 - M_{01}M_{02}M_{10} + M_{02}M_{12}M_{21}$$

$$r_6 = M_{02}M_{21}d_0 - M_{01}M_{02}M_{20} + M_{01}M_{12}M_{21}$$

$$r_7 = -M_{02}M_{10}d_2 + M_{01}M_{10}M_{12} - M_{02}M_{12}M_{20}$$

$$r_8 = M_{12}d_0d_1 - M_{02}M_{10}d_1 + M_{01}M_{10}M_{12} - M_{12}^2M_{21}$$

$$r_9 = M_{12}d_0d_1 - M_{02}M_{10}d_0 + M_{02}M_{12}M_{20} - M_{12}^2M_{21}$$

$$r_{10} = M_{01}d_1d_2 - M_{02}M_{21}d_2 + M_{01}M_{02}M_{20} - M_{01}^2M_{10}$$

$$r_{11} = M_{01}d_1d_2 + M_{02}M_{21}d_1 + M_{01}M_{12}M_{21} - M_{01}^2M_{10}$$

$$r_{12} = -M_{02}d_0d_2 + M_{01}M_{12}d_0 + M_{02}M_{12}M_{21} - M_{02}^2M_{20}$$

$$r_{13} = M_{02}d_0d_2 + M_{01}M_{12}d_2 - M_{01}M_{02}M_{10} + M_{02}^2M_{20}$$

$$r_{14} = d_0d_1d_2 - M_{01}M_{10}d_0 + M_{02}M_{20}d_1 - M_{12}M_{21}d_2$$

$$\mathbf{r} = (r_1, \dots, r_{14})$$

return $\mathbf{r}$

end function

Algorithm 8 implements the sum-of-products formula (76) for evaluating Δ . In addition to the Cauchy–Binet factorization, it incorporates the computation of diagonal differences to improve numerical stability near $J_2 = 0$, similar to the stable algorithms for J_2 and J_3 . Note that the individual factors r_i in the auxiliary vectors $\mathbf{u}$ and $\mathbf{v}$ contain the diagonal elements of the matrix $\mathbf{A}$ only in the form of their differences.

Lemma 7.2. *Any deviatoric backward stable algorithm for evaluating Δ must be forward stable in the sense that the absolute forward error must satisfy*

$$|\text{fl}(\Delta) - \Delta| \leq C \|\text{dev}(12J_2^2\mathbf{A}^\top - 54J_3 \text{cof}(\text{dev}(\mathbf{A})))\| \|\text{dev}(\mathbf{A})\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2). \quad (77)$$

Proof. This follows directly from the backward error analysis and Eq. (74). $\square$

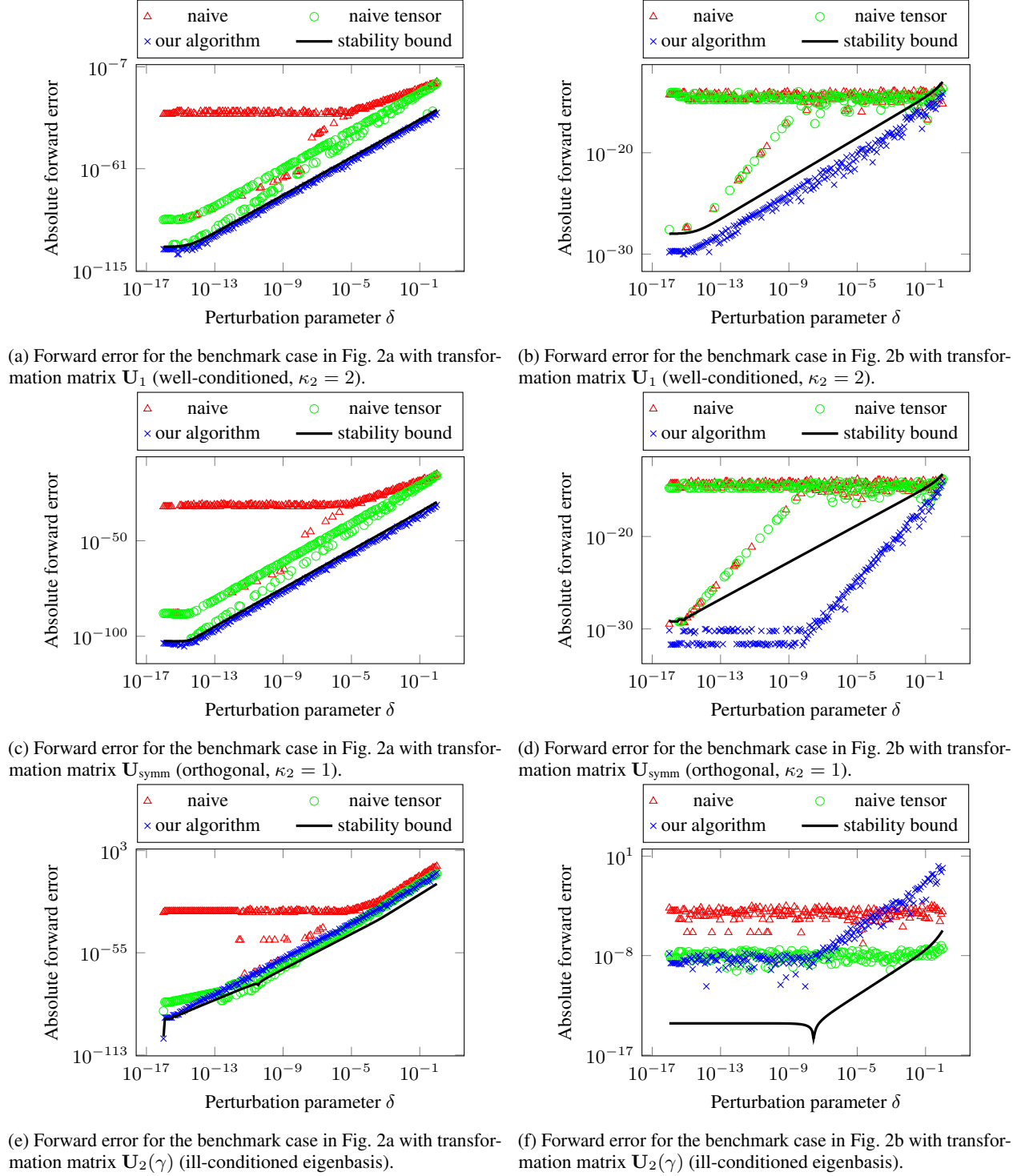
7.1 Discussion of numerical benchmarks

Numerical benchmarks evaluating the absolute forward error of Algorithm 8 are shown in Fig. 5.

Similar to the J_2 and J_3 invariants, we benchmark three implementations for evaluating Δ in Fig. 5. *Naive* and *naive tensor* implementations are based on the direct evaluation of the formula $\Delta = 4J_2^3 - 27J_3^2$, where J_2 and J_3 are computed using the *naive* and *naive tensor* algorithms, respectively.

The *naive* implementation is clearly unstable in all benchmark cases, with the forward error exceeding the stability bound by several orders of magnitude. The *naive tensor* implementation is more stable, but still fails to achieve the stability bound.

The proposed algorithm (Algorithm 8) is stable for benchmarks with a well-conditioned eigenbasis (Figs. 5a and 5b) and an orthogonal eigenbasis (Figs. 5c and 5d). In these cases, the forward error is close to the stability bound. However, the


 Figure 5: Numerical stability analysis for the discriminant Δ .

forward error increases for the benchmark with an ill-conditioned eigenbasis (Figs. 5e and 5f), exceeding the stability bound, and even those produced by the *naive* and *naive tensor* algorithms.

Note that the included stability bound plots (solid lines) in Fig. 5 are based only on the lowest order term from Lemma 7.2, i.e., $\|\text{dev}(12J_2^2 \mathbf{A}^\top - 54J_3 \text{cof}(\text{dev}(\mathbf{A})))\|_F \|\text{dev}(\mathbf{A})\|_F \epsilon_{\text{mach}}$. It could be shown that the higher order terms are negligible compared to the lowest order term in all benchmark cases.

7.2 Eigenvalues

For eigenvalues, there exists a classical perturbation result called the Bauer–Fike theorem (Bauer and Fike, 1960), which provides an absolute bound on the perturbation of eigenvalues of diagonalizable matrices.

Theorem 7.3 (Bauer–Fike, 1960). *Let $\mathbf{A} \in \mathbb{C}^{n \times n}$ be diagonalizable, i.e., $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{U}^{-1}$, where $\mathbf{D}$ is diagonal and $\mathbf{U}$ is invertible. Let λ be an eigenvalue of $\mathbf{A}$. Then there exists an eigenvalue $\tilde{\lambda}$ of $\mathbf{A} + \delta\mathbf{A}$ such that*

$$|\tilde{\lambda} - \lambda| \leq \kappa_p(\mathbf{U}) \|\delta\mathbf{A}\|_p, \quad (78)$$

where $\kappa_p(\mathbf{U}) = \|\mathbf{U}\|_p \|\mathbf{U}^{-1}\|_p$ is the p -norm condition number of the eigenbasis.

Proof. See, e.g., (Golub and Van Loan, 2013, Thm. 7.2.2). $\square$

We use the Bauer–Fike theorem to derive a stability bound for eigenvalue computation, summarized in the following lemma.

Lemma 7.4. *Any backward stable algorithm for evaluating the eigenvalues of a real diagonalizable matrix $\mathbf{A} \in \mathbb{R}^{3 \times 3}$ with real spectrum must be forward stable in the sense that the absolute forward error must satisfy*

$$|\text{fl}(\lambda) - \lambda| \leq C \kappa_2(\mathbf{U}) \|\mathbf{A}\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2), \quad (79)$$

where λ is an eigenvalue of $\mathbf{A}$, $\mathbf{U}$ is the eigenbasis of $\mathbf{A}$, and C is a moderate constant.

Proof. This is a consequence of the Bauer–Fike theorem, Theorem 7.3, and the definition of backward stability, similar to Theorem 5.3 and Lemmas 6.2 and 7.2.

Backward stability implies that there exists a perturbation $\delta\mathbf{A}$ such that

$$\text{fl}(\lambda(\mathbf{A})) = \lambda(\mathbf{A} + \delta\mathbf{A}), \quad \text{with} \quad \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|} \leq C \epsilon_{\text{mach}}. \quad (80)$$

Using the Bauer–Fike theorem with, for example, $p = 2$, we have

$$\begin{aligned} |\text{fl}(\lambda) - \lambda| &= |\lambda(\mathbf{A} + \delta\mathbf{A}) - \lambda(\mathbf{A})| \\ &\leq \kappa_2(\mathbf{U}) \|\delta\mathbf{A}\|_2 + \mathcal{O}(\|\delta\mathbf{A}\|) \\ &\leq C \kappa_2(\mathbf{U}) \|\mathbf{A}\| \epsilon_{\text{mach}} + \mathcal{O}(\epsilon_{\text{mach}}^2), \end{aligned} \quad (81)$$

where norm equivalence justifies the final transition to an arbitrary matrix norm $\|\cdot\|$. $\square$

Having developed stable algorithms for the invariants I_1, J_2, J_3 , and Δ , we can now propose an algorithm for the eigenvalues based on the trigonometric formula Eq. (2) and arctan expression for the triple-angle φ , Eq. (4), summarized in Algorithm 9.

Algorithm 9 Evaluation of the eigenvalues

Require: I_1, J_2, J_3 and Δ

$$t = \sqrt{27\Delta}/(27J_3)$$

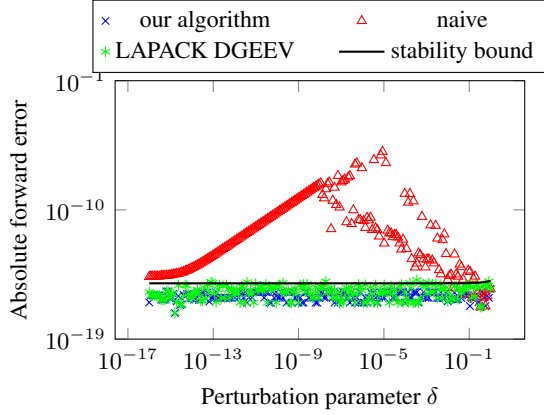
$$\varphi = \arctan(t)$$

$$\lambda_k = \frac{1}{3} (I_1 + 2\sqrt{3J_2} \cos((\varphi + 2\pi k)/3)) \text{ for } k \in \{1, 2, 3\}$$

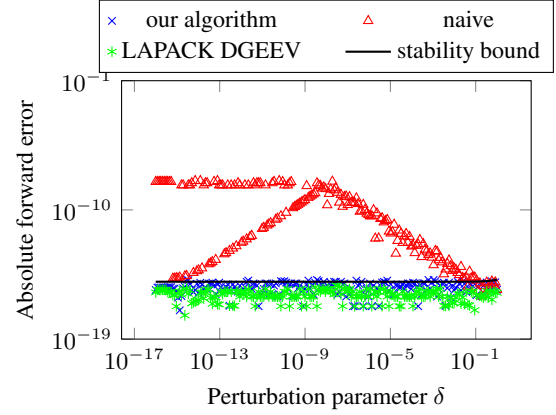
return $\lambda_1, \lambda_2, \lambda_3$

▷ Triple-angle argument

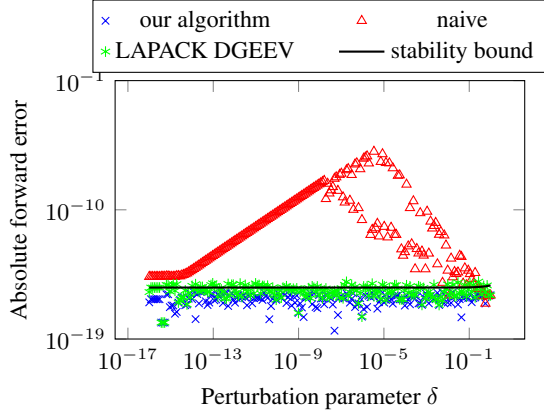
▷ Triple-angle



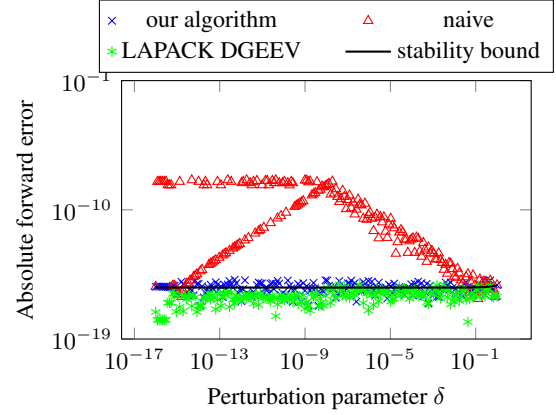
(a) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



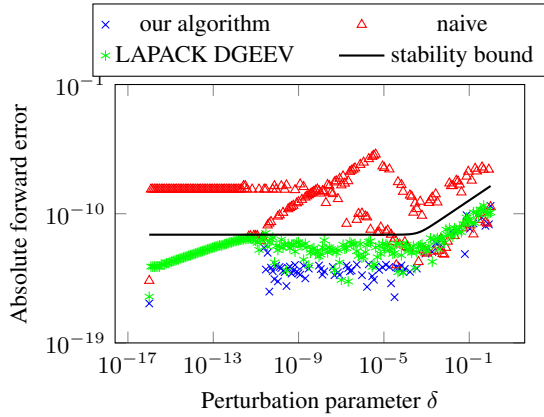
(b) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_1$ (well-conditioned, $\kappa_2 = 2$).



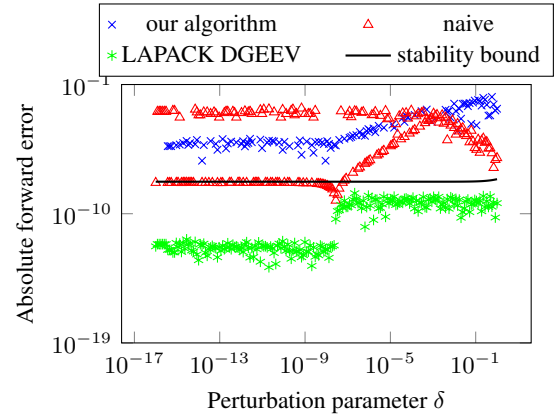
(c) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(d) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_{\text{symm}}$ (orthogonal, $\kappa_2 = 1$).



(e) Forward error for the benchmark case in Fig. 2a with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).



(f) Forward error for the benchmark case in Fig. 2b with transformation matrix $\mathbf{U}_2(\gamma)$ (ill-conditioned eigenbasis).

Figure 6: Numerical stability analysis for eigenvalue computation.

7.3 Discussion of numerical benchmarks

Three different implementations of eigenvalue evaluation are benchmarked in Fig. 6: *naive*, *LAPACK DGEEV*, and *our algorithm*.

Naive approach is based on Algorithm 9 with invariants J_2 , J_3 and Δ computed with the naive algorithms, i.e., Algorithm 3 and Algorithm 6.

LAPACK DGEEV is based on the LAPACK library routine DGEEV, which computes all eigenvalues and, optionally, the left and/or right eigenvectors of a real nonsymmetric matrix (Anderson et al., 1999). We use the NumPy wrapper `numpy.linalg.eigvals`, see (NumPy, 2025b), for this routine.

The *naive* approach is numerically unstable and produces forward errors as large as $\epsilon_{\text{mach}}^{1/3} \approx 10^{-5}$ for the benchmark case of a nearly triple eigenvalue in Figs. 6a, 6c, and 6e. For the benchmark case of a nearly double eigenvalue in Figs. 6b, 6d, and 6f, the forward error is as large as $\epsilon_{\text{mach}}^{1/2} \approx 10^{-8}$.

For well-conditioned cases with $\mathbf{U} = \mathbf{U}_1$ (Figs. 6a and 6b) and orthogonal cases with $\mathbf{U} = \mathbf{U}_{\text{symm}}$ (Figs. 6c and 6d), *our algorithm* satisfies the stability bound from Lemma 7.4.

LAPACK DGEEV produces forward errors that are close to the stability bound in all benchmark cases, even in the most challenging case of the transformation matrix being nonorthogonal and nearly singular, $\mathbf{U} = \mathbf{U}_2(\gamma)$ (Figs. 6e and 6f). The γ parameter was chosen as $\gamma = 10^{-3}$, which leads to condition number $\kappa_2(\mathbf{U}_2) \approx 9 \times 10^3$.

8 Performance benchmarks

In this section, we present performance benchmarks of the proposed eigenvalue algorithm in comparison with the numerical library LAPACK (Anderson et al., 1999). The benchmarks were executed on a MacBook Pro (2024) with Apple M4 (ARM) CPU (10-core CPU, 120 GB/s memory bandwidth).

The benchmarks were written in C11 with LAPACK routine DGEEV called via the LAPACKE C interface version 3.12.1. The OpenBLAS library version 0.3.29 was linked for BLAS and LAPACK functionality. This setup was run inside a Docker container based on the official Python 3.14 Docker image `python:3.14-trixie`. The container is pre-installed with GCC version 14.2.0. The code was compiled with optimization flags `-O3 -march=native`.

The benchmark consists of evaluating eigenvalues of an example real, diagonalizable 3×3 matrix. The matrix was generated as in Eq. (17) with transformation matrix $\mathbf{U} = \mathbf{U}_1$ (well-conditioned case, $\kappa_2(\mathbf{U}_1) = 2$) and eigenvalues along the benchmark path in Fig. 2a, i.e., $\mathbf{D} = \text{diag}(-1, 1, 1 + 10^{-14})$. This is a challenging case of nearly a double eigenvalue, but both J_2 and J_3 remain finite and away from zero. The test matrix reads explicitly as

$$\mathbf{A} = \mathbf{U}_1 \mathbf{D} \mathbf{U}_1^{-1} = \begin{bmatrix} \text{fl}(0) & \text{fl}(5 \cdot 10^{-15}) & \text{fl}(1 + 5 \cdot 10^{-15}) \\ \text{fl}(-1) & \text{fl}(1 + 5 \cdot 10^{-15}) & \text{fl}(1 + 5 \cdot 10^{-15}) \\ \text{fl}(1) & \text{fl}(5 \cdot 10^{-15}) & \text{fl}(5 \cdot 10^{-15}) \end{bmatrix}. \quad (82)$$

We performed a total of 10^6 evaluations for the above matrix and measured the total execution time for both the proposed algorithm and LAPACK DGEEV. The results are presented in Table 1. The proposed algorithm is approximately ten times faster than LAPACK DGEEV for this benchmark, while both methods returned eigenvalues with the expected absolute forward error on the order of machine precision, i.e., approximately 10^{-16} .

Table 1: Performance comparison of eigenvalue computation for the test matrix $\mathbf{A}$ over 10^6 evaluations.

Method	Average time per evaluation $\pm$ std. [ns]	Fastest time per evaluation [ns]
Our algorithm	38.2 ± 1.2	35.02
LAPACK DGEEV	396.4 ± 4.3	381.18

For convenience, we also provide wrappers for Python using CFFI (Rigo and Fijalkowski, 2025). Our implementation is available as part of the open-source library `eig3x3`, see (Habera and Zilian, 2025). The library `eig3x3` contains implementations of all algorithms presented in this paper, including naive and stable algorithms for evaluating the invariants J_2 , J_3 , and Δ , as well as eigenvalue computation and the benchmarking code used to generate the results in this paper. The eigenvalue algorithms are available as

- `eig3x3.eigvals` for computing the eigenvalues of a real, diagonalizable 3×3 matrix,
- `eig3x3.eigvalss` for computing the eigenvalues of a real, symmetric 3×3 matrix.

Note that the C implementation of the proposed algorithm is not optimized for the specific CPU architecture. Further optimizations, such as SIMD vectorization, could lead to even better performance. On the other hand, LAPACK is a highly optimized library that benefits from years of development and architecture-specific tuning.

The proposed algorithm is closed-form and can be inlined in performance-critical code sections, while LAPACK routines typically involve function-call overhead. This can further widen the performance gap in favor of the proposed algorithm in practical scenarios.

9 Conclusion

In this work, we have presented a detailed numerical stability analysis of closed-form expressions for the eigenvalues of 3×3 real matrices. We have focused on the computation of four key invariants: the trace I_1 , the deviatoric invariants J_2 and J_3 , and the discriminant Δ . For each invariant, we derived forward error bounds and proposed specialized algorithms designed to be stable, particularly in the challenging cases of coalescing eigenvalues.

Our analysis and numerical benchmarks demonstrate that the proposed algorithm for the invariant J_2 is accurate, satisfying the derived stability bounds even for matrices with ill-conditioned eigenbases. The algorithms for J_3 and the discriminant Δ , however, are stable for matrices with well-conditioned or orthogonal eigenbases but their accuracy degrades significantly for matrices with ill-conditioned eigenbases, where they fail to meet the theoretical stability bounds. The final eigenvalue computation, which relies on these invariants, inherits their stability characteristics. Consequently, the proposed closed-form solution is numerically stable and accurate for matrices with a well-conditioned eigenbasis, significantly outperforming naive implementations. For the most challenging cases involving ill-conditioned eigenbases, the established iterative library routine LAPACK DGEEDV provides more accurate results.

Performance benchmarks show that the proposed closed-form algorithm is approximately ten times faster than the highly optimized LAPACK implementation for a challenging test case with nearly double eigenvalue. This highlights the potential of closed-form solutions in performance-critical applications, especially considering that our implementation is not fully optimized and could be inlined to avoid function call overhead.

The header-only open-source library eig3x3 (Habera and Zilian, 2025) provides the C implementations of the proposed algorithms with a thin Python interface (via CFFI), including eigenvalue routines for both general and symmetric 3×3 matrices: eig3x3.eigvals (real, diagonalizable) and eig3x3.eigvalss (real, symmetric). The repository includes naive and stable variants for J_2 , J_3 , and Δ , together with build scripts and benchmarks to reproduce the results reported here.

Future work should focus on proving the forward stability of invariants J_3 and Δ (and consequently the eigenvalues), which we currently only observe empirically for well-conditioned eigenbases. Further performance gains could be achieved by architecture-specific optimizations, such as vectorization. In conclusion, while iterative methods remain the gold standard for accuracy in the most ill-conditioned problems, our work provides a robust and efficient closed-form alternative for a large and practical class of matrices.

References

- Anderson, E. et al. (1999). *LAPACK Users' Guide*. Third. Philadelphia, PA: Society for Industrial and Applied Mathematics. ISBN: 0-89871-447-8 (paperback).
- Bauer, F. L. and C. T. Fike (Dec. 1960). "Norms and exclusion theorems". In: *Numerische Mathematik* 2.1, pp. 137–141. ISSN: 0945-3245. DOI: 10.1007/bf01386217. URL: <http://dx.doi.org/10.1007/BF01386217>.
- Blinn, James F. (May 2007). "How to Solve a Cubic Equation, Part 5: Back to Numerics". In: *IEEE Computer Graphics and Applications* 27.3, pp. 78–89. ISSN: 1558-1756. DOI: 10.1109/mcg.2007.60. URL: <http://dx.doi.org/10.1109/MCG.2007.60>.
- Bronstein, I.N. et al. (2015). *Handbook of Mathematics*. Springer Berlin Heidelberg. ISBN: 9783662462218. DOI: 10.1007/978-3-662-46221-8. URL: <http://dx.doi.org/10.1007/978-3-662-46221-8>.
- Golub, Gene H. and Charles F. Van Loan (2013). *Matrix Computations - 4th Edition*. Philadelphia, PA: Johns Hopkins University Press. DOI: 10.1137/1.9781421407944. eprint: <https://epubs.siam.org/doi/pdf/10.1137/1.9781421407944>. URL: <https://epubs.siam.org/doi/abs/10.1137/1.9781421407944>.
- Habera, Michal and Andreas Zilian (2021). *Symbolic spectral decomposition of 3x3 matrices*. DOI: 10.48550/ARXIV.2111.02117. URL: <https://arxiv.org/abs/2111.02117>.
- (2025). *Eigenvalues of 3x3 matrices*. URL: <https://github.com/michalhabera/eig3x3> (visited on 01/10/2025).

- Harari, Isaac and Uri Albocher (Nov. 2022). “Computation of eigenvalues of a real, symmetric 3×3 matrix with particular reference to the pernicious case of two nearly equal eigenvalues”. In: *International Journal for Numerical Methods in Engineering* 124.5, pp. 1089–1110. ISSN: 1097-0207. DOI: 10.1002/nme.7153. URL: <http://dx.doi.org/10.1002/nme.7153>.
- (July 2023). “Using the discriminant in a numerically stable symmetric 3×3 direct eigenvalue solver”. In: *International Journal for Numerical Methods in Engineering* 124.20, pp. 4473–4489. ISSN: 1097-0207. DOI: 10.1002/nme.7311. URL: <http://dx.doi.org/10.1002/nme.7311>.
- Harris, Charles R. et al. (Sept. 2020). “Array programming with NumPy”. In: *Nature* 585.7825, pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: <https://doi.org/10.1038/s41586-020-2649-2>.
- Higham, Nicholas J. (Jan. 2002). *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics. ISBN: 9780898718027. DOI: 10.1137/1.9780898718027. URL: <http://dx.doi.org/10.1137/1.9780898718027>.
- La Porte, M (1973). “Une Formulation Numériquement Stable Donnant les Racines Réelles de l’Équation du 3^{ème} Degré”. In: *IFP Report* 21516.
- Fredrik Johansson, et al. (2023). *mpmath: a Python library for arbitrary-precision floating-point arithmetic (version 1.3.0)*. <http://mpmath.org/>.
- NumPy (2025a). *numpy.linalg.det*. NumPy v2.3 Manual. URL: <https://numpy.org/doc/2.3/reference/generated/numpy.linalg.det.html> (visited on 01/10/2025).
- (2025b). *numpy.linalg.eigvals*. NumPy v2.3 Manual. URL: <https://numpy.org/doc/2.3/reference/generated/numpy.linalg.eigvals.html> (visited on 01/10/2025).
- (2025c). *numpy.matmul*. NumPy v2.3 Manual. URL: <https://numpy.org/doc/2.3/reference/generated/numpy.matmul.html> (visited on 01/10/2025).
- (2025d). *numpy.trace*. NumPy v2.3 Manual. URL: <https://numpy.org/doc/2.3/reference/generated/numpy.trace.html> (visited on 01/10/2025).
- Parlett, Beresford N. (Nov. 2002). “The (matrix) discriminant as a determinant”. In: *Linear Algebra and its Applications* 355.1–3, pp. 85–101. ISSN: 0024-3795. DOI: 10.1016/S0024-3795(02)00335-X. URL: [http://dx.doi.org/10.1016/S0024-3795\(02\)00335-X](http://dx.doi.org/10.1016/S0024-3795(02)00335-X).
- Press, William H. et al. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. USA: Cambridge University Press. ISBN: 0521880688.
- Rigo, Armin and Maciej Fijalkowski (2025). *CFFI documentation*. C Foreign Function Interface for Python. Version 2.0.0. URL: <https://cffi.readthedocs.io/en/stable/> (visited on 10/17/2025).
- Scherzinger, W.M. and C.R. Dohrmann (Aug. 2008). “A robust algorithm for finding the eigenvalues and eigenvectors of 3×3 symmetric matrices”. In: *Computer Methods in Applied Mechanics and Engineering* 197.45–48, pp. 4007–4015. ISSN: 0045-7825. DOI: 10.1016/j.cma.2008.03.031. URL: <http://dx.doi.org/10.1016/j.cma.2008.03.031>.
- Smith, Oliver K (1961). “Eigenvalues of a symmetric 3×3 matrix”. In: *Communications of the ACM* 4.4, p. 168.
- Trefethen, Lloyd N. and David Bau (Jan. 1997). *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics. ISBN: 9780898719574. DOI: 10.1137/1.9780898719574. URL: <http://dx.doi.org/10.1137/1.9780898719574>.