PACF: A precision-adjustable computational framework for solving singular values[1]

Chuanying Li[a,b], Roberto Barrio[c], Xiong Xiao[a,b], Peibing Du[d], Hao Jiang[e], Zhe Quan[a,b,*], Kenli Li[a,b]

[a] *College of Computer Science and Electronic Engineering, Hunan University, China*
[b] *National Supercomputing Center in Changsha, Hunan University, China*
[c] *Departamento de Matemática Aplicada and IUMA, University of Zaragoza, Spain*
[d] *Northwest Institute of Nuclear Technology, China*
[e] *College of Computer, National University of Defense Technology, China*

Abstract

Singular value decomposition (SVD) plays a significant role in matrix analysis, and the differential quotient difference with shifts (DQDS) algorithm is an important technique for solving singular values of upper bidiagonal matrices. However, ill-conditioned matrices and large-scale matrices may cause inaccurate results or long computation times when solving singular values. At the same time, it is difficult for users to effectively find the desired solution according to their needs. In this paper, we design a precision-adjustable computational framework for solving singular values, named PACF. In our framework, the same solution algorithm contains three options: original mode, high-precision mode, and mixed-precision mode. The first algorithm is the original version of the algorithm. The second algorithm is a reliable numerical algorithm we designed using Error-free transformation (EFT) technology. The last algorithm is an efficient numerical algorithm we developed using the mixed-precision idea. Our PACF can add different solving algorithms for different types of matrices, which are universal and extensible. Users can choose different algorithms to solve singular values according to different needs. This paper implements the high-precision DQDS and mixed-precision DQDS algorithms and conducts extensive experiments on a supercomputing platform to demonstrate that our algorithm is reliable and efficient. Besides, we introduce the error analysis of the inner loop of the DQDS and HDQDS algorithms.

Keywords: DQDS algorithm, Singular value, Error-free transformation, Mixed-precision, Error analysis

[1] Post-print article published in: Applied Mathematics and Computation, Volume 440, 127611,(2023).

[*] Corresponding author
Email address: quanzhe@hnu.edu.cn (Zhe Quan)

1. Introduction

Singular value decomposition (SVD) is widely used in scientific research, urban construction, medical services, and other issues in daily life, such as rolling element bearing diagnosis [1], image de-noising and compression [2], analysis of phase-space growth rates [3], enhanced resolution of seismic data [4], image analysis using PDEs [5], and so on. It is a significant tool of linear algebra in matrix analysis. There are various methods for calculating matrix singular values [6, 7], and these methods have important practical significance. For example, matrix singular values play an essential role in the least-squares method. However, we need to use different algorithms when solving singular values of different types of matrices. This is relatively time-consuming work. Besides, there will be some problems when solving the singular value of the matrix, such as inaccurate results and long calculation time.

Generally, the inaccuracy of the solution results is owing to the existence of rounding errors. And rounding errors are not avoided in floating point arithmetic [8, 9]. Rounding errors can lead to inaccurate or even wrong calculation results after massive accumulation, especially when encountering ill-conditioned problems. Error-free transformations (EFT) technology [10] shows reliable performance in overcoming the cumulative of reducing rounding errors. The key idea is to return the rounding error of operations to the original calculation result through cumulative compensation [11, 12]. For another aspect, the mixed-precision idea can be used to reduce computation time while maintaining the same number of program iterations [13, 14, 15]. Numerical methods using this idea generally use higher precision in the algorithm's key calculation steps and lower precision in other parts. It takes advantage of the lower precision formats' high performance and reduces the overhead of global computation by adopting mixed data formats.

Based on this, we propose a precision-adjustable multi-mode computational framework named PACF, used in this article for solving singular values. Users can choose different algorithms to solve singular values according to different needs. The same solution algorithm contains three modes in our framework: original mode, high-precision mode, and mixed-precision mode. The first is the original version of the algorithm, the second is a reliable numerical algorithm we designed using EFT technology, and the last is an efficient numerical algorithm we designed using the mixed-precision idea. Our PACF can add different solving algorithms for different types of matrices, which are universal and extensible. Currently, we implement the differential quotient difference with shifts (DQDS) algorithm [16, 17] that calculates the singular values of upper bidiagonal matrices. We will introduce the DQDS algorithm as an example in the subsequent content. We conduct extensive experimental verifications on a supercomputing platform to demonstrate the effectiveness of the above ideas. The five main contributions of this article are shown below.

- We design a high-precision DQDS algorithm based on EFT technology, which makes the solution results more reliable. We further propose a

double-double format DQDS algorithm to better illustrate the HDQDS algorithm's accuracy.

- We devise a mixed-precision DQDS algorithm based on the idea of mixed-precision, exploiting the potential performance advantages of different precision formats to reduce overhead.
- We propose a precision-adjustable computational framework for solving singular values, which can provide different modes according to different matrices and different solution requirements.
- We describe the error analysis of the inner loop of the DQDS algorithm and the high-precision DQDS algorithm.
- Extensive experiments verify that our designed algorithm has a superior performance to the original algorithm, and our framework is reliable and efficient.

The remainder of this work is organized as below. We provide a background overview and comments on related literature in Section 2. Section 3 introduces DQDS algorithms, rounding errors, and EFT technology. In Section 4, we exhaustively describe our framework and proposed algorithm. Section 5 describes the error analysis of the inner loop of the DQDS and HDQDS algorithm. Section 6 shows the accuracy and stability of our algorithm through numerical experiments. Section 7 summarizes the whole paper as well as introduces future work.

2. Related Work

This section mainly introduces the related background knowledge of the DQDS algorithm (Section 2.1), EFT technique (Section 2.2), and mixed-precision (Section 2.3).

2.1. The DQDS Algorithm

The DQDS algorithm was designed by Parlett *et al.* [16] after adding the shift s based on the differential quotient difference (DQD) algorithm. It has been shown that the existence of the shift s accelerates the convergence of the algorithm. Aishima *et al.* presented some convergence theorems of the DQDS algorithm [18], and illustrated them with experiments. Besides, they gave the specific steps of shift strategy of the DQDS algorithm, and rigorously proved that this strategy has asymptotic cubic convergence [19]. Aishima et al. [20, 21] introduced its corresponding convergence rate theorem for the DQDS algorithm under different shift strategies. Furthermore, Aishima et al. [22] proposed a new DQDS method in subsequent work, and the asymptotic super quadratic convergence was realized. Li *et al.* [23] also improved DQDS, and they designed a novel deflation strategy that can significantly improve the calculation speed

of the DQDS algorithm. LAPACK-3.4.1 has adopted this strategy to enhance the DQDS algorithm.

Nakatsukasa *et al.* [24] combined DQDS with aggressive early deflation technology, which greatly decreases the running cost of DQDS. Araki *et al.* [25] combined DQDS and OQDS (orthogonal QD with shift) to calculate the column space of the rectangular matrix, and used the DQDS method to study the distribution of all singular values to determine the numerical rank. Ferreira and Parlett *et al.* [26] proposed a new DQDS algorithm to calculate the eigenvalues of the real tridiagonal matrix, and presented experiments to show that the new transform triple DQDS is faster. Many researchers [27] applied the Kato-Temple inequality to the matrix eigenvalues, and then proposed a new shift strategy. However, most of these studies are aimed at shift strategy and convergence analysis. It does not consider the rounding errors' accumulation in the calculation, resulting in a decrease in the accuracy of the calculation, and the problem of excessive CPU time when calculating large-scale matrices. Previous work [28] only considered rounding errors in DQDS algorithms, and just proposed high-precision DQDS algorithms. In this paper, we present the error analysis of the inner loop of the DQDS and HDQDS algorithms, implement the DQDS algorithm in double-double format, and propose an MDQDS algorithm based on the idea of mixed-precision to reduce overhead while ensuring global accuracy. We further propose a generic precision-adjustable computational framework for solving singular values.

2.2. EFT Technique

In floating point arithmetic systems, rounding errors will inevitably reduce the numerical algorithms' accuracy, so increasing the algorithms' accuracy is a reliable method to improve the quality of numerical simulations. Error-free transformations (EFT) were proposed by Ogita *et al.* [10] in 2005, and it is a powerful mathematical strategy to reduce rounding errors in numerical calculations. The key idea is to return the rounding error of operations to the original calculation result through cumulative compensation. Du *et al.* [8] proposed a Compensated QD algorithm-COMPQD algorithm based on EFT. The COMPQD algorithm records the error generated by each step of the operation and compensates for the original result to make the result more accurate.

For matrix multiplication, fusion with EFT can develop reliable numerical algorithms. Ozaki [9] described it in detail and analyzed the rounding error. Barrio *et al.* [29] designed ORTHOPOLY software using EFT techniques to effectively and accurately evaluate a series of classical orthogonal polynomials and their derivatives. Graillat *et al.* [12] verified the compensated algorithms by using stochastic arithmetic, and then proved the accuracy of the compensated algorithm using EFT. Jiang *et al.* [11] proposed the compensated Horner derivative algorithm, which is used to accurately evaluate the k-th derivative of a polynomial in power basis. And the accuracy of the algorithm is proved by experiments. In this paper, we design a high-precision DQDS algorithm, which combines the DQDS algorithm with EFT. It improves the accuracy of calculation by reducing the influence of rounding errors.

2.3. Mixed-precision Strategy

The reason single-precision computations are faster than double-precision computations is that lower precision can perform more operations per second on traditional processors [30]. Some researchers switch the algorithm to single precision in order to speed up, and most of this operation is feasible. But in some cases, single-precision computations can lead to non-convergence. Thus mixed-precision is developed. Mixed-precision is to use different precisions in different parts of an algorithm according to requirements, such as half-precision, single-precision, double-precision, which improve the algorithm's performance by combining data formats [31]. For example, mixed-precision versions of linear algebra algorithms are greatly accelerated in scientific computing [13]. The GMRES algorithm for solving sparse systems of linear equations utilizes mixed-precision ideas to achieve over 8% speedup [14].

Basic Linear Algebra Subroutines (BLAS) design a mixed-precision version, which implements more accurate and faster algorithms, and it is a typical mixed-precision library [32]. Yamazaki *et al.* [33] designed a mixed-precision version of the Cholesky QR algorithm, improving numerical reliability in practical applications. Higham *et al.* [34] developed the multi-precision Schur–Parlett algorithm, which has the same reliability as the original algorithm. Sun *et al.* [15] designed the mixed-precision linear solver, which improved the overall performance by changing the data format. Mixed-precision is widely used in many scenarios, such as reducing training time in artificial intelligence [35]. Based on this, we propose a mixed-precision DQDS algorithm, which is accelerated by the potential performance advantages of mixed-precision.

3. Preliminaries

In this section, we introduce the basics of the DQDS algorithm and related algorithms of the rounding errors and EFT techniques.

3.1. DQDS Algorithm: Basics

The DQDS algorithm, proposed by Fernando and Parlett in 1994 [16], can efficiently calculate the bidiagonal matrices' singular values and is also used to calculate the tridiagonal matrices' eigenvalues. The shift strategy of the DQDS algorithm has been deeply studied in the literature, so it has a faster convergence speed when used for large-scale matrices. Parlett and Marques described in detail how to efficiently implement the DQDS algorithm in Ref. [17]. At this time, DQDS has been included in the LAPACK library as an open-source algorithm and is widely used as a DLASQ routine.

Now, we will briefly introduce the DQDS algorithm. Firstly, we have to transform the input matrix A to the upper bidiagonal matrix B by successively

applying the known Householder transforms before executing the DQDS algorithm, obtaining

$$B = \begin{pmatrix} b_1 & b_2 & & \\ & b_3 & \ddots & \\ & & \ddots & b_{2k-2} \\ & & & b_{2k-1} \end{pmatrix}. \tag{1}$$

In the following we assume that we begin the algorithm from the upper bidiagonal matrix B and that all non-zero elements in B are positive, $b_t > 0$ for $t = 1, 2, ..., 2k-1$.

The DQDS algorithm [17] adds variable d_k compared with QDS (orthogonal QD) [36], and $d_k (k = 1, .., n-1) > 0$, but it has the advantage of compensation because all variables in DQDS must be non-negative numbers.

In order to better describe the DQDS algorithm, we use the following format for the upper bidiagonal matrix

$$B^{(n)} = \begin{pmatrix} \sqrt{q_1^{(n)}} & \sqrt{e_1^{(n)}} & & \\ & \sqrt{q_2^{(n)}} & \ddots & \\ & & \ddots & \sqrt{e_{m-1}^{(n)}} \\ & & & \sqrt{q_m^{(n)}} \end{pmatrix} \tag{2}$$

According to the Cholesky transform with shift, the DQDS algorithm can be written as:

$$(B^{(n+1)})^{\mathrm{T}} B^{(n+1)} = B^{(n)} (B^{(n)})^{\mathrm{T}} - s^{(n)} I, \tag{3}$$

where $s^{(n)} \geq 0$ $(n = 0, 1, 2, ...,)$ is the shift, and I is the unit matrix. The specific process of the DQDS algorithm is shown in Algorithm 1 [17].

Algorithm 1 The DQDS Algorithm

Input:

$B^{(n)}$ $(n = 0, 1, 2, 3, ...)$

Output:

$q_i^{(n)}$ $(i = 1, 2, 3, ..., m)$

$e_i^{(n)}$ $(i = 1, 2, 3, ..., m-1)$

1: Initialization:

$q_k^{(0)} = (b_{2k-1})^2$ $(k = 1, 2, 3, ..., m)$

$e_k^{(0)} = (b_{2k})^2$ $(k = 1, 2, 3, ..., m-1)$

2: **for** $n = 0, 1, 2, 3, ...$ **do**

3: choose shift $s^{(n)} (\geq 0)$

4: $d_1^{(n+1)} = q_1^{(n)} - s^{(n)}$

5: **for** $k = 1, 2, 3, ...m-1$ **do**

$q_k^{(n+1)} = d_k^{(n+1)} + e_k^{(n)}$
$e_k^{(n+1)} = e_k^{(n)} q_{k+1}^{(n)} / q_k^{(n+1)}$
$d_{k+1}^{(n+1)} = (d_k^{(n+1)} q_{k+1}^{(n)} / q_k^{(n+1)}) - s^{(n)}$
end for
$q_m^{(n+1)} = d_m^{(n+1)}$
if convergence criterion **STOP**
end for

When the convergence criterion set by the experiment is suitable, the outer loop will end, such as $|e_{m-1}^{(n)}| \leq \xi$, where ξ>0 is a constant. The diagonal elements of the matrix are arranged in descending order of its singular values when the DQDS algorithm ends the loop. DQDS is similar to the Cholesky LR method with shifts when applied to tridiagonal symmetric matrices. It should be noted that the DQDS algorithm shows excellent performance in calculating singular values and has other interesting applications. Different shift strategies can be selected according to different problems, and in that case, the convergence rate will also change. Ref.[20](Chapter 4) comprehensively introduces various shift strategies and their convergence rate.

3.2. Rounding Errors and EFT

The rounding error is usually expressed as the difference between the approximate calculated values during floating point operations and exact values. It will cause the unreliability of the target value calculation result. Especially when calculating massive data, the accumulation of a large number of rounding errors will cause the calculation result to deviate from the expected result. However, rounding errors in floating point operations are unavoidable. Recently, Yang *et al.* [37] conducted a detailed analysis of the mixed-precision algorithm's rounding error, and developed a mixed-precision version of the Householder QR algorithm. Therefore, a large number of scholars have been studying rounding errors.

As the accuracy requirements for floating point calculations are getting higher and higher, the IEEE organization expanded the IEEE-754(1985) standard in 2008 to address the problem of inaccurate computing results caused by a large number of rounding errors. They supplemented quadruple precision (128 bit) and decimal arithmetic to meet current computing needs, thus forming a new set of arithmetic standards IEEE-754 (2008).

A standard floating point calculation model in floating point operations [38] is:

$$f_1 \; op \; f_2 = fl(f_1 \circ f_2) = (f_1 \circ f_2)(1 + \varepsilon_1) = (f_1 \circ f_2)/(1 + \varepsilon_2), \tag{4}$$

where $\circ \in \{+, -, \times, \div\}$, $|\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}$, $f_1, f_2 \in \mathbb{R}$, *op* represents the addition, subtraction, multiplication, and division operations inside the computer during the execution of floating point calculations, fl is expressed as a floating point

operation. $\mathbf{u}$ is the unit roundoff. We define [38]

$$<N> := 1 + \theta_N = \prod_{n=1}^{N} (1 + \varepsilon_n)^{\rho_n}, \tag{5}$$

where $|\varepsilon_n| \leq \mathbf{u}, \rho_n = \pm 1$,

$$|\theta_n| \leq \gamma_n := \frac{nu}{(1 - nu)}, \tag{6}$$

for $n = 1, 2, \ldots, N$ and $nu < 1$.

Under the current working accuracy, the design of a compensated algorithm can effectively reduce the rounding error. EFT is the core tool of our compensated algorithm. Suppose now floating point numbers $(f_1, f_2) \in \mathbb{F}$ ($\mathbb{F}$ denotes the floating point numbers set), $\circ \in \{+, -, \times\}$, and $z = fl(f_1 \circ f_2) \in \mathbb{F}$. Also, we assume rounding to nearest without overflow and underflow. Then the definition of EFT is:

$$(f_1 \circ f_2) = z + err, \tag{7}$$

where z is the best floating point approximation of the computing result, and err is the exact rounding error.

Now we describe the basic EFT algorithms for addition, product and division of floating point numbers. Suppose $(f_1, f_2) \in \mathbb{F}$, the sum of f_1 and f_2 is denoted s, and the rounding error is s_err. EFT of the sum of two floating point numbers are:

$$\begin{aligned} &f_1 \oplus f_2 = s, \\ &f_1 \ominus s \oplus f_2 = s\_err, \end{aligned} \tag{8}$$

and

$$\begin{aligned} &f_1 \oplus f_2 = s, \\ &t = s \ominus f_1, \\ &s\_err = (f_1 \ominus (s \ominus t)) \oplus (f_2 \ominus t). \end{aligned} \tag{9}$$

Eqns.(8) and (9) are the `FastTwoSum` algorithm [39] and the `TwoSum` algorithm [40], respectively. We remark that the `FastTwoSum` algorithm requires $|f_1| \geq |f_2|$.

The `TwoProd` algorithm [39] is the EFT of two floating-point numbers' products. The Fused-Multiply-and-Add (FMA) instruction [41] is almost as fast in execution time as multiplication or addition, improving numerical precision by reducing rounding. Hence, we obtain the `TwoProdFMA` algorithm by combining FMA with the `TwoProd` algorithm, that is:

$$\begin{aligned} &f_1 \otimes f_2 = p, \\ &FMA(f_1, f_2, -p) = p\_err. \end{aligned} \tag{10}$$

with $(f_1, f_2) \in \mathbb{F}$, p represent the result of product of floating point numbers, p_err stands for rounding error.

The `DivRem` algorithm [42, 43] is the EFT of the division of two floating point numbers. Similarly, the input to the algorithm is $(f_1, f_2) \in \mathbb{F}$, the output is r and r_err, r is the result of the division of floating point numbers, r_err stands for rounding error,

$$\begin{aligned} &f_1 \oslash f_2 = r, \\ &r \otimes f_2 = p, \\ &FMA(r, f_2, -p) = p\_err, \\ &(f_1 \ominus p) \ominus p\_err = r\_err. \end{aligned} \tag{11}$$

4. Numerical Framework

We propose a precision-adjustable computational framework for solving singular values. First, we introduce the overall design of our proposed framework in Section 4.1, and then we take DQDS as an example to introduce the high-precision DQDS algorithm in Section 4.2, the double-double version in Section 4.3, and the mixed-precision DQDS algorithm in Sections 4.4, respectively.

4.1. Overview of the PACF

This section describes a multi-mode framework in which users can choose different algorithms to solve singular values according to different precision needs. The core strategy of our framework is that the same algorithm contains three modes: original mode, high-precision mode, and mixed-precision mode. The original version of the algorithm is the original mode. The high-precision mode is a reliable numerical algorithm designed by us using EFT technology to reduce the accumulation of rounding errors. The mixed-precision mode is an efficient numerical algorithm designed by carefully combining the data format using the mixed-precision idea. As shown in Fig.1, our framework can add different solving algorithms for different types of problems. According to our idea, the newly added algorithms can be transformed into three precision modes, so our framework is universal and extensible.

Fig.1 shows the flow of the PACF framework. Firstly, we input the matrix to be solved and select the corresponding algorithm according to the matrix type. Then we select the corresponding mode according to your precision needs. For example, you have to choose a high-precision algorithm if you need high calculation accuracy, and to choose a mixed-precision algorithm if you need a fast calculation speed. Finally, output the calculation result.

Now, we introduce the DQDS algorithm for solving singular values of upper bidiagonal matrices as an example. In literature, there are plenty of studies on the shift strategy of the DQDS algorithm. On the contrary, rounding errors have generally been ignored in the solution procedure as the DQDS algorithm is considered stable. However, it can lead to slightly inaccurate solution results because rounding errors are unavoidable in floating point calculations for very

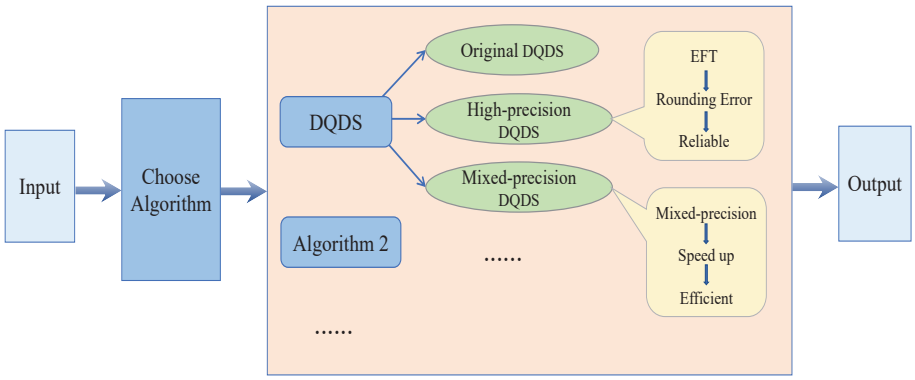

Figure 1: Overview of the PACF.

large matrices. One of our main goals is to develop roundoff accurate solutions, and not just stable ones. The EFT technique we use can effectively reduce the accumulation of rounding errors. It compensates the rounding error generated at each step of the DQDS algorithm, and it allows us to design a reliable numerical algorithm, named HDQDS. We describe the high-precision DQDS algorithm (HDQDS) in detail in Section 4.2. Moreover, we take into account the performance differences of computations in different precision formats, and utilize the potential performance advantages of mixed precision to improve the performance of numerical algorithms. Numerical methods that use mixed-precision ideas generally use higher precision (u_{high}) in the key calculation steps of the algorithm, and lower precision (u_{low}) in other parts. Therefore, we reduce the overall computational overhead by carefully combining the data formats with utilizing the speedup of the lower precision format. We developed an efficient numerical algorithm, named MDQDS. Section 4.4 details the mixed-precision DQDS algorithm. Whether it is a high-precision DQDS algorithm or a mixed-precision DQDS algorithm, their inputs and outputs have the same format as the original DQDS, which is double precision.

4.2. A High-precision DQDS Algorithm

In this section, we introduce a high-precision DQDS algorithm (HDQDS), which can also be called a compensated DQDS algorithm. We present the notations in Table 1, also used in the error analysis, for a better understanding of this paper.

Table 1: Notations used on the different DQDS algorithms and in the error analysis.

Notations	Description
b_{2k-1}	Element in a bidiagonal matrix
$q_k^{(n)}$	The exact value
$\widetilde{q}_k^{(n)}$	Result with perturbed inputs in real arithmetic
$\widehat{q}_m^{(n)}$	Result of computation in floating point operations
$\varsigma\widehat{q}_k^{(n)}$	The compensated term of $\widehat{q}_k^{(n)}$
$\delta\widetilde{q}_k^{(n)}$	The perturbation term of $\widetilde{q}_k^{(n)}$
$\varrho\widehat{q}_k^{(n)}$	The approximate rounding errors of $\widehat{q}_k^{(n)}$
$\varrho\widetilde{q}_k^{(n)}$	The approximate perturbation term for $\widetilde{q}_k^{(n)}$
$\varsigma\varrho\widehat{q}_k^{(n)}$	The compensated term of $\varrho\widehat{q}_k^{(n)}$
$s^{(n)}$	The shift value of each iteration of the algorithm

First of all, we perform some preliminary steps for the initialization of the DQDS algorithm, as the initial data is supposed to be calculated previously, and it is an approximate value rather than an exact one due to the rounding errors. We utilize the `TwoProdFMA` algorithm to make the original error be discarded to enter in the subsequent calculations.

Next, we start to consider each inner loop of DQDS, record the rounding error generated by each step of the operation through EFT, and then compensate back to the original calculation result. Then, a more accurate value will be obtained.

For the first inner loop

$$[\widehat{d}_1^{(n+1)}, \varrho\widehat{d}_1^{(n+1)}] = \texttt{TwoSum}(\widehat{q}_1^{(n)}, -\widehat{s}^{(n)}), \tag{12}$$

we can deduce the compensated term $\varrho\widehat{d}_1^{(n+1)}$ of $\widehat{d}_1^{(n+1)}$

$$\varrho\widehat{d}_1^{(n+1)} = \varrho\widehat{d}_1^{(n+1)} \oplus \varrho\widehat{q}_1^{(n)}. \tag{13}$$

Now, we can use the `FastTwoSum` algorithm to update $\widehat{d}_1^{(n+1)}$ and $\varrho\widehat{d}_1^{(n+1)}$, and so the updated $\widehat{d}_1^{(n+1)}$ is more accurate than before

$$[\widehat{d}_1^{(n+1)}, \varrho\widehat{d}_1^{(n+1)}] = \texttt{FastTwoSum}(\widehat{d}_1^{(n+1)}, \varrho\widehat{d}_1^{(n+1)}). \tag{14}$$

For the second inner loop

$$\begin{aligned} &[\widehat{q}_k^{(n+1)}, \nu_1] = \texttt{TwoSum}(\widehat{d}_k^{(n+1)}, \widehat{e}_k^{(n)}), \\ &[t, \nu_2] = \texttt{TwoProdFMA}(\widehat{e}_k^{(n)}, \widehat{q}_{k+1}^{(n)}), \\ &[\widehat{e}_k^{(n+1)}, \nu_3] = \texttt{DivRem}(t, \widehat{q}_k^{(n+1)}). \end{aligned} \tag{15}$$

And so, we can get the approximate compensation term $\varrho\widehat{q}_k^{(n+1)}$ of $\widehat{q}_k^{(n+1)}$

$$\varrho\widehat{q}_k^{(n+1)} = \varrho\widehat{q}_k^{(n+1)} \oplus \varrho\widehat{d}_k^{(n+1)} \oplus \varrho\widehat{e}_k^{(n)}. \tag{16}$$

When calculating $\varrho\widehat{e}_k^{(n+1)}$ with perturbed input, according to the formula, we obtain

$$\begin{aligned}\varrho\widehat{e}_k^{(n+1)} =&(\varrho\widehat{e}_k^{(n)} \otimes \widehat{q}_{k+1}^{(n)} \oplus \widehat{e}_k^{(n)} \otimes \varrho\widehat{q}_{k+1}^{(n)} \ominus \varrho\widehat{q}_k^{(n+1)} \otimes \widehat{e}_k^{(n+1)} \ominus \nu_2 \otimes \widehat{e}_k^{(n)} \\ &\ominus \nu_3 \otimes \widehat{q}_k^{(n+1)}) \oslash \widehat{q}_k^{(n+1)}.\end{aligned} \tag{17}$$

The calculation process of $\widehat{d}_k^{(n+1)}$ should be divided into three steps, the first step is

$$\begin{aligned}&[t, \nu_5] = \texttt{TwoProdFMA}(\widehat{d}_k^{(n+1)}, \widehat{q}_{k+1}^{(n)}), \\ &[\widehat{\mu}, \nu_6] = \texttt{DivRem}(t, \widehat{q}_k^{(n+1)}).\end{aligned} \tag{18}$$

Then, computing the compensated term in (12), we obtain

$$\begin{aligned}\varrho\widehat{\mu} =&(\varrho\widehat{d}_k^{(n+1)} \otimes \widehat{q}_{k+1}^{(n)} \oplus \widehat{d}_k^{(n+1)} \otimes \varrho\widehat{q}_{k+1}^{(n)} \ominus \varrho\widehat{q}_k^{(n+1)} \otimes \widehat{\mu} \ominus \nu_5 \otimes \widehat{d}_k^{(n+1)} \\ &\ominus \nu_6 \otimes \widehat{q}_k^{(n+1)}) \ominus \widehat{q}_k^{(n+1)}.\end{aligned} \tag{19}$$

We update μ and $\varrho\widehat{\mu}$ by using the `FastTwoSum` algorithm, and then we use the `TwoSum` algorithm to calculate $d_{k+1}^{(n+1)}$, that is

$$[\widehat{d}_{k+1}^{(n+1)}, \varrho\widehat{d}_{k+1}^{(n+1)}] = \texttt{TwoSum}(\mu, -\widehat{s}^{(n)}). \tag{20}$$

Finally, the compensated term we obtain is

$$\varrho\widehat{d}_{k+1}^{(n+1)} = \varrho\widehat{d}_{k+1}^{(n+1)} \oplus \varrho\widehat{\mu}. \tag{21}$$

For each step of the DQDS algorithm's inner loop, we utilize the `FastTwoSum` algorithm to update the calculated results and compensated terms. The updated results are more accurate than the original result. Therefore, the input of the high-precision DQDS has not changed compared with the original DQDS. However, the intermediate calculation process uses EFT to reduce rounding errors and make the final result more accurate. Note that we do not compute rounding errors for the shift s to make our framework more extensible. Users can choose the shift strategy we provide according to the problem to be solved or add it themselves. We propose a new high-precision DQDS algorithm based on the above discussion, as shown in Algorithm 2.

Algorithm 2 The High-precision DQDS Algorithm (HDQDS)

Input:
$\widehat{B}^{(n)}$ $(n = 0, 1, 2, 3, ...)$
Output:
$\widehat{q}_i^{(n)}$ $(i = 1, 2, 3, ..., m)$
$\widehat{e}_i^{(n)}$ $(i = 1, 2, 3, ..., m-1)$

Initialization:
$[\widehat{q}_k^{(0)}, \widehat{\varrho q}_k^{(0)}] = \texttt{TwoProdFMA}(\widehat{b}_{2k-1}, \widehat{b}_{2k-1}) \quad (k = 1, 2, 3, ..., m)$
$[\widehat{e}_k^{(0)}, \widehat{\varrho e}_k^{(0)}] = \texttt{TwoProdFMA}(\widehat{b}_{2k}, \widehat{b}_{2k}) \quad (k = 1, 2, 3, ..., m-1)$
for $n = 0, 1, 2, 3, ...$ **do**
  choose shift $\widehat{s}^{(n)} (\geq 0)$
  $[\widehat{d}_1^{(n+1)}, dt] = \texttt{TwoSum}(\widehat{q}_1^{(n)}, -\widehat{s}^{(n)})$
  $\widehat{\varrho d}_1^{(n+1)} = dt + \widehat{\varrho q}_1^{(n)}$
  $[\widehat{d}_1^{(n+1)}, \widehat{\varrho d}_1^{(n+1)}] = \texttt{FastTwoSum}(\widehat{d}_1^{(n+1)}, \widehat{\varrho d}_1^{(n+1)})$
  for $k = 1, 2, 3, ...m-1$ **do**
    $[\widehat{q}_k^{(n+1)}, \nu_1] = \texttt{TwoSum}(\widehat{d}_k^{(n+1)}, \widehat{e}_k^{(n)})$
    $\widehat{\varrho q}_k^{(n+1)} = \nu_1 \oplus \widehat{\varrho d}_k^{(n+1)} \oplus \widehat{\varrho e}_k^{(n)}$
    $[\widehat{q}_k^{(n+1)}, \widehat{\varrho q}_k^{(n+1)}] = \texttt{FastTwoSum}\ (\widehat{q}_k^{(n+1)}, \widehat{\varrho q}_k^{(n+1)})$
    $[t, \nu_2] = \texttt{TwoProdFMA}(\widehat{e}_k^{(n)}, \widehat{q}_{k+1}^{(n)})$
    $[\widehat{e}_k^{(n+1)}, \nu_3] = \texttt{DivRem}(t, \widehat{q}_k^{(n+1)})$
    $\widehat{\varrho e}_k^{(n+1)} = (\widehat{\varrho e}_k^{(n)} \otimes \widehat{q}_{k+1}^{(n)} \oplus \widehat{e}_k^{(n)} \otimes \widehat{\varrho q}_{k+1}^{(n)} \ominus \widehat{\varrho q}_k^{(n+1)} \otimes \widehat{e}_k^{(n+1)} \ominus \nu_2 \otimes \widehat{e}_k^{(n)}$
    $\ominus \nu_3 \otimes \widehat{q}_k^{(n+1)}) \oslash \widehat{q}_k^{(n+1)}$
    $[\widehat{e}_k^{(n+1)}, \widehat{\varrho e}_k^{(n+1)}] = \texttt{FastTwoSum}(\widehat{e}_k^{(n+1)}, \widehat{\varrho e}_k^{(n+1)})$
    $[t, \nu_5] = \texttt{TwoProdFMA}(\widehat{d}_k^{(n+1)}, \widehat{q}_{k+1}^{(n)})$
    $[\widehat{\mu}, \nu_6] = \texttt{DivRem}(t, \widehat{q}_k^{(n+1)})$
    $\widehat{\varrho \mu} = (\widehat{\varrho d}_k^{(n+1)} \otimes \widehat{q}_{k+1}^{(n)} \oplus \widehat{d}_k^{(n+1)} \otimes \widehat{\varrho q}_{k+1}^{(n)} \ominus \widehat{\varrho q}_k^{(n+1)} \otimes \widehat{\mu} \ominus \nu_5 \otimes \widehat{d}_k^{(n+1)}$
    $\ominus \nu_6 \otimes \widehat{q}_k^{(n+1)}) \ominus \widehat{q}_k^{(n+1)}$
    $[\widehat{\mu}, \widehat{\varrho \mu}] = \texttt{FastTwoSum}(\widehat{\mu}, \widehat{\varrho \mu})$
    $[\widehat{d}_{k+1}^{(n+1)}, \nu_4] = \texttt{TwoSum}(\widehat{\mu}, -\widehat{s}^{(n)})$
    $\widehat{\varrho d}_{k+1}^{(n+1)} = \nu_4 \oplus \widehat{\varrho \mu}$
    $[\widehat{d}_{k+1}^{(n+1)}, \widehat{\varrho d}_{k+1}^{(n+1)}] = \texttt{FastTwoSum}(\widehat{d}_{k+1}^{(n+1)}, \widehat{\varrho d}_{k+1}^{(n+1)})$
  end for
  $\widehat{q}_m^{(n+1)} = \widehat{d}_m^{(n+1)}$
  if convergence criterion **STOP**
end for

4.3. A Double-double Format DQDS Algorithm

To better explain the accuracy of the HDQDS algorithm, we implement a double-double format DQDS algorithm, named DD_DQDS algorithm, and it is presented in Algorithm 3. The basic algorithms of double-double format (`prod_dd_dd`, `add_dd_dd`, `Div_dd_dd`) used in DD_DQDS are detailed in Refs.[32, 42, 44, 45, 46]. Moreover, the introduction of this extra precision algorithm allows us to compare the accuracy and running time of DQDS, HDQDS, and DD_DQDS algorithms (see Section 6.2).

Algorithm 3 The DD_DQDS Algorithm

Input:
$Bh^{(n)}, Bl^{(n)} \quad (n = 0, 1, 2, 3, ...)$

Output:

$qh_i^{(n)} \quad (i = 1, 2, 3, ..., m)$

$eh_i^{(n)} \quad (i = 1, 2, 3, ..., m-1)$

1: Initialization:

$[qh_k^{(0)}, ql_k^{(0)}] = \texttt{prod\_dd\_dd}(bh_{2k-1}, bl_{2k-1}, bh_{2k-1}, bl_{2k-1}) \quad (k = 1, 2, 3, ..., m)$

$[eh_k^{(0)}, el_k^{(0)}] = \texttt{prod\_dd\_dd}(bh_{2k}, bl_{2k}, bh_{2k}, bl_{2k}) \quad (k = 1, 2, 3, ..., m-1)$

2: **for** $n = 0, 1, 2, 3, ...$ **do**

3: choose shift $s^{(n)} (\geq 0)$

4: $[dh_1^{(n+1)}, dl_1^{(n+1)}] = \texttt{add\_dd\_dd}(qh_1^{(n)}, ql_1^{(n)}, -sh^{(n)}, -sl^{(n)})$

5: **for** $k = 1, 2, 3, ...m-1$ **do**

6: $[qh_k^{(n+1)}, ql_k^{(n+1)}] = \texttt{add\_dd\_dd}(dh_k^{(n+1)}, dl_k^{(n+1)}, eh_k^{(n)}, el_k^{(n)})$

7: $[vh, vl] = \texttt{Div\_dd\_dd}(qh_{k+1}^{(n)}, ql_{k+1}^{(n)}, qh_k^{(n+1)}, ql_k^{(n+1)})$

8: $[eh_k^{(n+1)}, el_k^{(n+1)}] = \texttt{prod\_dd\_dd}(eh_k^{(n)}, el_k^{(n)}, vh, vl)$

9: $[temph, templ] = \texttt{prod\_dd\_dd}(dh_k^{(n+1)}, dl_k^{(n+1)}, vh, vl)$

10: $[dh_{k+1}^{(n+1)}, dl_{k+1}^{(n+1)}] = \texttt{add\_dd\_dd}(temph, templ, -sh^{(n)}, -sl^{(n)})$

11: **end for**

12: $qh_m^{(n+1)} = dh_m^{(n+1)}$

$ql_m^{(n+1)} = dl_m^{(n+1)}$

13: if convergence criterion **STOP**

14: **end for**

4.4. A Mixed-precision DQDS Algorithm

In this section, we introduce the mixed-precision DQDS algorithm for fast solving singular values in Algorithm 4. The core idea of mixed-precision algorithms is iterative refinement, which efficiently utilizes available computing power by carefully combining data formats.

The transmission, processing, and storage of data in the computer are binary. In the IEEE-754 standard, data in double-precision format occupies 64 bits, and single-precision format is 32 bits. Fig.2 shows the representation of π in single- and double-precision floating point formats. Although double-precision arithmetic will compute more accurate results, 64-bit floating point operations (double) are typically half as fast as 32-bit floating point operations (float). To effectively take advantage of the performance differences of calculations under different precision formats, we adopt a mixed data format by using different floating point formats in different operations to ensure accuracy while reducing the overall overhead and accelerating calculation. We demonstrate the effectiveness of our MDQDS algorithm by conducting experiments on a supercomputing platform. In addition, we uniformly compute the shift s with double precision to make our framework extensible. Users can choose the shift strategy we provide according to the problem or add it.

Our strategy is to use full-precision in the algorithm's key computational steps and reduced-precision for other parts. In this article, u_{high} represents double-precision computing, u_{low} represents single-precision computing, and the

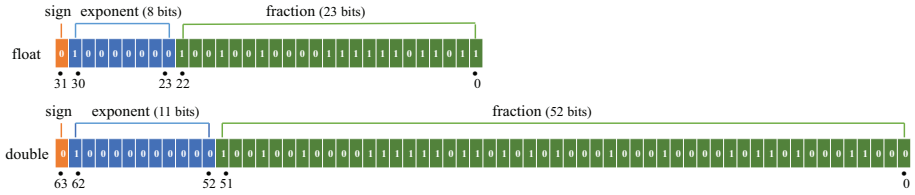

Figure 2: Difference between single- and double-precision floating point formats of π.

subscripts $_{high}$ and $_{low}$ mean that the precision format of the data is double and float, respectively.

In Algorithm 4 we detail the MDQDS algorithm. Specifically, the addition and subtraction of two floating point numbers are calculated using double-precision, and the multiplication and division of two floating point numbers are calculated using single-precision, but the result is stored in double-precision format. For example, we use double-precision regardless of the calculation or storage in lines 4 and 6, but for line 7 we use single-precision when calculating and store the result in double precision.

Algorithm 4 The MDQDS Algorithm

Input:

$B^{(n)} \quad (n = 0, 1, 2, 3, ...)$

Output:

$q_i^{(n)} \quad (i = 1, 2, 3, ..., m)$

$e_i^{(n)} \quad (i = 1, 2, 3, ..., m-1)$

1: Initialization:

$(q_k^{(0)})_{high} = ((b_{2k-1})_{low})^2 \quad (k = 1, 2, 3, ..., m)$

$(e_k^{(0)})_{high} = ((b_{2k})_{low})^2 \quad (k = 1, 2, 3, ..., m-1)$

2: **for** $n = 0, 1, 2, 3, ...$ **do**

3: choose shift $(s^{(n)})_{high} (\geq 0)$

4: $(d_1^{(n+1)})_{high} = (q_1^{(n)})_{high} - (s^{(n)})_{high}$

5: **for** $k = 1, 2, 3, ...m-1$ **do**

6: $(q_k^{(n+1)})_{high} = (d_k^{(n+1)})_{high} + (e_k^{(n)})_{high}$

7: $(e_k^{(n+1)})_{high} = (e_k^{(n)})_{low} (q_{k+1}^{(n)})_{low} / (q_k^{(n+1)})_{low}$

8: $(d_{k+1}^{(n+1)})_{high} = (d_k^{(n+1)})_{low} (q_{k+1}^{(n)})_{low} / (q_k^{(n+1)})_{low} - (s^{(n)})_{high}$

9: **end for**

10: $(q_m^{(n+1)})_{high} = (d_m^{(n+1)})_{high}$

11: if convergence criterion **STOP**

12: **end for**

5. Error Analysis

In this section, we present the error analysis of the inner loop of the DQDS algorithm (see Section 5.1) and the high-precision DQDS algorithm (see Section 5.2). In order to better understand the following content, we list the meaning of the symbols in Table 1.

5.1. Error Analysis of the DQDS Algorithm

In this subsection, we analyze the main loop of the DQDS Algorithm. In the second inner loop of DQDS (lines 5-9 of the Algorithm 1), we discuss the floating point inputs' perturbations, that is

$$\begin{aligned} \widehat{q}_k^{(n)} &= q_k^{(n)} + \varsigma\widehat{q}_k^{(n)}, \\ \widehat{e}_k^{(n)} &= e_k^{(n)} + \varsigma\widehat{e}_k^{(n)}, \\ \widehat{d}_k^{(n)} &= d_k^{(n)} + \varsigma\widehat{d}_k^{(n)}. \end{aligned} \tag{22}$$

Therefore, line 6 of the algorithm ($q_k^{(n+1)} = d_k^{(n+1)} + e_k^{(n)}$) will satisfy

$$\widetilde{q}_k^{(n+1)} = \widehat{d}_k^{(n+1)} + \widehat{e}_k^{(n)}, \tag{23}$$

and line 7 ($e_k^{(n+1)} = e_k^{(n)} q_{k+1}^{(n)} / q_k^{(n+1)}$)

$$\widetilde{e}_k^{(n+1)} = \widehat{e}_k^{(n)} \widehat{q}_{k+1}^{(n)} / \widehat{q}_k^{(n+1)}, \tag{24}$$

and finally, line 8 ($d_{k+1}^{(n+1)} = d_k^{(n+1)} q_{k+1}^{(n)} / q_k^{(n+1)} - s^{(n)}$)

$$\widetilde{d}_{k+1}^{(n+1)} = \widehat{d}_k^{(n+1)} \widehat{q}_{k+1}^{(n)} / \widehat{q}_k^{(n+1)} - \widehat{s}^{(n)}. \tag{25}$$

In Eqs.(23), (24) and (25), all the calculations are executed using real arithmetic without rounding error. But, if all the calculations are executed in floating-point operation, we get

$$\begin{aligned} fl(\widetilde{q}_k^{(n+1)}) = \widehat{q}_k^{(n+1)} &= \widehat{d}_k^{(n+1)} \oplus \widehat{e}_k^{(n)}, \\ fl(\widetilde{e}_k^{(n)}) = \widehat{e}_k^{(n+1)} &= fl(fl(\widehat{e}_k^{(n)} \widehat{q}_{k+1}^{(n)}) / \widehat{q}_k^{(n+1)}), \\ fl(\widetilde{d}_{k+1}^{(n+1)}) = \widehat{d}_{k+1}^{(n+1)} &= fl(fl(\widehat{d}_k^{(n+1)} \widehat{q}_{k+1}^{(n)}) / \widehat{q}_k^{(n+1)}) \ominus \widehat{s}^{(n)}. \end{aligned} \tag{26}$$

Next, we give the absolute perturbation bounds for the DQDS algorithm's inner loop when the input is a floating point number (22).

Lemma 1. *The absolute perturbation bounds for the DQDS algorithm's inner loop when using real arithmetic and input floating point numbers, are given by*

$$|\delta\widetilde{q}_k^{(n+1)}| \leq |\varsigma\widehat{d}_k^{(n+1)}| + |\varsigma\widehat{e}_k^{(n)}|, \tag{27}$$

$$|\delta\widetilde{e}_k^{(n+1)}| \leq \beta_k^{(n+1)}, \tag{28}$$

and

$$|\delta\widehat{d}_{k+1}^{(n+1)}| \leq \lambda_k^{(n+1)} + |\varsigma\widehat{s}^{(n)}|, \tag{29}$$

where

$$\beta_k^{(n+1)} = b_k^{(n+1)} \times \frac{|e_k^{(n)}||\varsigma\widehat{q}_{k+1}^{(n)}| + |\varsigma\widehat{e}_k^{(n)}||q_{k+1}^{(n)}| + |e_k^{(n+1)}||\varsigma\widehat{q}_k^{(n+1)}| + |\varsigma\widehat{e}_k^{(n)}||\varsigma\widehat{q}_{k+1}^{(n)}|}{|q_k^{(n+1)}|}, \tag{30}$$

$$\lambda_k^{(n+1)} = b_k^{(n+1)} \times \frac{|d_k^{(n+1)}||\varsigma\widehat{q}_{k+1}^{(n)}| + |\varsigma\widehat{d}_k^{(n+1)}||q_{k+1}^{(n)}| + |d_{k+1}^{(n+1)}||\varsigma\widehat{q}_k^{(n+1)}| + |\varsigma\widehat{d}_k^{(n+1)}||\varsigma\widehat{q}_{k+1}^{(n)}|}{|q_k^{(n+1)}|}, \tag{31}$$

with

$$b_k^{(n+1)} = |\frac{q_k^{(n+1)}}{q_k^{(n+1)} + \varsigma\widehat{q}_k^{(n+1)}}|, \tag{32}$$

assuming $q_k^{(n+1)} \neq 0$ *and* $\frac{\varsigma\widehat{q}_k^{(n+1)}}{q_k^{(n+1)}} \neq -1$.

Proof 1. *From (22) and (23), we get that*

$$\delta\widetilde{q}_k^{(n+1)} = \varsigma\widehat{d}_k^{(n+1)} + \varsigma\widehat{e}_k^{(n)}, \tag{33}$$

which give us (27).

In a similar way, we have

$$\delta\widetilde{e}_k^{(n+1)} = \frac{e_k^{(n)}\varsigma\widehat{q}_{k+1}^{(n)} + \varsigma\widehat{e}_k^{(n)}q_{k+1}^{(n)} - e_k^{(n+1)}\varsigma\widehat{q}_k^{(n+1)} + \varsigma\widehat{e}_k^{(n)}\varsigma\widehat{q}_{k+1}^{(n)}}{q_k^{(n+1)} + \varsigma\widehat{q}_k^{(n+1)}} \tag{34}$$

and

$$\delta\widetilde{d}_{k+1}^{(n+1)} = \frac{d_k^{(n+1)}\varsigma\widehat{q}_{k+1}^{(n)} + \varsigma\widehat{d}_k^{(n+1)}q_{k+1}^{(n)} + d_{k+1}^{(n+1)}\varsigma\widehat{q}_k^{(n+1)} + \varsigma\widehat{d}_k^{(n+1)}\varsigma\widehat{q}_{k+1}^{(n)}}{q_k^{(n+1)} + \varsigma\widehat{q}_k^{(n+1)}} - s^{(n)}. \tag{35}$$

Therefore, we get the bound (28) and (29) if $q_k^{(n+1)} \neq 0$ *and* $\frac{\varsigma\widehat{q}_k^{(n+1)}}{q_k^{(n+1)}} \neq -1$, *where* $\beta_k^{(n+1)}$ *and* $\lambda_k^{(n+1)}$ *are defined in (30) and (31), respectively.* ⊠

We now give the perturbation analysis of the DQDS algorithm, and then we analyze the rounding error in floating-point operations. We assume that all inputs are floating point numbers without any perturbations.

Lemma 2. *Let* $fl(\widetilde{q}_k^{(n+1)}) = \widehat{q}_k^{(n+1)}$, $fl(\widetilde{e}_k^{(n+1)}) = \widehat{e}_k^{(n+1)}$ *and* $fl(\widetilde{d}_{k+1}^{(n+1)}) = \widehat{d}_{k+1}^{(n+1)}$ *are calculated in floating-point operations in the inner loop of DQDS algorithm, thus*

$$|fl(\widetilde{q}_k^{(n+1)}) - \widehat{q}_k^{(n+1)}| \leq \gamma_2(|\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)}|), \tag{36}$$

$$|fl(\widetilde{e}_k^{(n+1)}) - \widetilde{e}_k^{(n+1)}| \leq \gamma_2 |\widetilde{e}_k^{(n+1)}|, \tag{37}$$

and

$$|fl(\widetilde{d}_{k+1}^{(n+1)}) - \widetilde{d}_{k+1}^{(n+1)}| \leq \gamma_2 |\widetilde{d}_{k+1}^{(n+1)}|. \tag{38}$$

Proof 2. *It is obtained directly from (4) and (26).* ⊠

We can deduce the rounding error of the DQDS algorithm's inner loop in floating point operations from Lemma 1 and 2.

Lemma 3. *When the input is a perturbed floating-point number, the rounding error bound for the inner loop of the DQDS algorithm is given by*

$$|fl(\widetilde{q}_k^{(n+1)}) - q_k^{(n+1)}| \leq \gamma_2(|d_k^{(n+1)}| + |e_k^{(n)}|) + (1+\gamma_2)(|\varsigma \widehat{d}_k^{(n+1)}| + |\varsigma \widehat{e}_k^{(n)}|), \tag{39}$$

$$|fl(\widetilde{e}_k^{(n+1)}) - e_k^{(n+1)}| \leq \gamma_2 |e_k^{(n+1)}| + (1+\gamma_2)\beta_k^{(n+1)}, \tag{40}$$

and

$$|fl(\widetilde{d}_{k+1}^{(n+1)}) - d_{k+1}^{(n+1)}| \leq \gamma_2 |d_{k+1}^{(n+1)}| + (1+\gamma_2)(\lambda_k^{(n+1)} - |\varsigma \widehat{s}^{(n)}|). \tag{41}$$

where $\beta_k^{(n+1)}$ and $\lambda_k^{(n+1)}$ are defined in (30) and (31).

Proof 3. *Firstly, we have*

$$|fl(\widetilde{q}_k^{(n+1)}) - q_k^{(n+1)}| \leq |fl(\widetilde{q}_k^{(n+1)}) - \widetilde{q}_k^{(n+1)}| + |\widetilde{q}_k^{(n+1)} - q_k^{(n+1)}|. \tag{42}$$

Next by (27) in Lemma 1, (36) in Lemma 2, and (22), we get

$$\begin{aligned} |fl(\widetilde{q}_k^{(n+1)}) - q_k^{(n+1)}| &\leq \gamma_2(|\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)}|) + (|\varsigma \widehat{d}_k^{(n+1)}| + |\varsigma \widehat{e}_k^{(n)}|) \\ &\leq \gamma_2(|d_k^{(n+1)}| + |e_k^{(n)}|) + (1+\gamma_2)(|\varsigma \widehat{d}_k^{(n+1)}| + |\varsigma \widehat{e}_k^{(n)}|). \end{aligned} \tag{43}$$

Finally, by (28) in Lemma 1 and (37) in Lemma 2, we have

$$\begin{aligned} |fl(\widetilde{e}_k^{(n+1)}) - e_k^{(n+1)}| &\leq |fl(\widetilde{e}_k^{(n+1)}) - \widetilde{e}_k^{(n+1)}| + |\widetilde{e}_k^{(n+1)} - e_k^{(n+1)}| \\ &\leq \gamma_2 |e_k^{(n+1)}| + (1+\gamma_2)\beta_k^{(n+1)}, \end{aligned} \tag{44}$$

Similarly, by (29) in Lemma 1 and (38) in Lemma 2, we obtain

$$\begin{aligned} |fl(\widetilde{d}_{k+1}^{(n+1)}) - d_{k+1}^{(n+1)}| &\leq |fl(\widetilde{d}_{k+1}^{(n+1)}) - \widetilde{d}_{k+1}^{(n+1)}| + |\widetilde{d}_{k+1}^{(n+1)} - d_{k+1}^{(n+1)}| \\ &\leq \gamma_2 |d_{k+1}^{(n+1)}| + (1+\gamma_2)(\lambda_{k+1}^{(n+1)} - |\varsigma \widehat{s}^{(n)}|). \end{aligned} \tag{45}$$

⊠

The above results show us that the error bounds for the DQDS algorithm's inner loop are of order γ_j and/or terms multiplied by compensated terms (that is, terms of the type $\varsigma\square$). Therefore, we have error bounds of order 1 in the roundoff unit and the DQDS algorithm is stable, but for large problems, as the number of operations increases, the terms that multiply the roundoff unit can grow and the accuracy decreases.

5.2. Error Analysis of the High-precision DQDS Algorithm

The below part performs error analysis on the main loop of the High-precision DQDS algorithm (lines 7-22 of Algorithm (2)).

Firstly, we take into account the floating point inputs' perturbations. Let

$$\varrho\widetilde{q}_k^{(n+1)} = \varrho\widehat{d}_k^{(n+1)} + \varrho\widehat{e}_k^{(n)} - \nu_1 \tag{46}$$

$$\varrho\widetilde{e}_k^{(n+1)} = \frac{\varrho\widehat{e}_k^{(n)}\widehat{q}_{k+1}^{(n)} + \widehat{e}_k^{(n)}\varrho\widehat{q}_{k+1}^{(n)} - \varrho\widehat{q}_k^{(n+1)}\widehat{e}_k^{(n+1)} - \varrho\widehat{e}_k^{(n)}\varrho\widehat{q}_{k+1}^{(n)} - \nu_2\widehat{e}_k^{(n)} - \nu_3\widehat{q}_k^{(n+1)}}{\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}} \tag{47}$$

$$\varrho\widetilde{d}_{k+1}^{(n+1)} = temp - \varrho\widehat{s}^{(n)} - \nu_4, \tag{48}$$

where

$$temp = \frac{\varrho\widehat{d}_k^{(n+1)}\widehat{q}_{k+1}^{(n)} + \widehat{d}_k^{(n+1)}\varrho\widehat{q}_{k+1}^{(n)} - \varrho\widehat{q}_k^{(n+1)}\widehat{temp} - \varrho\widehat{d}_k^{(n+1)}\varrho\widehat{q}_{k+1}^{(n)} - \nu_5\widehat{d}_k^{(n+1)} - \nu_6\widehat{q}_k^{(n+1)}}{\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}}. \tag{49}$$

Lemma 4. *The bound of $\varrho\widetilde{q}_k^{(n+1)}$ is given by*

$$|\varrho\widetilde{q}_k^{(n+1)} - \varsigma\widehat{q}_k^{(n+1)}| \leq |\varsigma\varrho\widehat{d}_k^{(n+1)}| + |\varsigma\varrho\widehat{e}_k^{(n)}|. \tag{50}$$

And the bounds of $\varrho\widetilde{e}_k^{(n+1)}$ and $\varrho\widetilde{d}_{k+1}^{(n+1)}$ are given by

$$|\varrho\widetilde{e}_k^{(n+1)} - \varsigma\widehat{e}_k^{(n+1)}| \leq \omega_k^{(n+1)}, \tag{51}$$

and

$$|\varrho\widetilde{d}_{k+1}^{(n+1)} - \varsigma\widehat{d}_{k+1}^{(n+1)}| \leq \chi_k^{(n+1)} + |\varsigma\varrho\widehat{s}^{(n)}|, \tag{52}$$

where

$$\omega_k^{(n+1)} = c_k^{(n+1)} \times \frac{|e_k^{(n)}||\varsigma\varrho\widehat{q}_{k+1}^{(n)}| + |\varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)}| + |e_k^{(n+1)}||\varsigma\varrho\widehat{q}_k^{(n+1)}| + |\varsigma\varrho\widehat{e}_k^{(n)}||\varsigma\varrho\widehat{q}_{k+1}^{(n)}|}{|q_k^{(n+1)}|}, \tag{53}$$

$$\chi_k^{(n+1)} = c_k^{(n+1)} \times \frac{|d_k^{(n+1)}||\varsigma\varrho\widehat{q}_{k+1}^{(n)}| + |\varsigma\varrho\widehat{d}_k^{(n+1)}||q_{k+1}^{(n)}| + |d_{k+1}^{(n+1)}||\varsigma\varrho\widehat{q}_k^{(n+1)}| + |\varsigma\varrho\widehat{d}_k^{(n+1)}||\varsigma\varrho\widehat{q}_{k+1}^{(n)}|}{|q_k^{(n+1)}|}, \tag{54}$$

with

$$c_k^{(n+1)} = |\frac{q_k^{(n+1)}}{q_k^{(n+1)} - \varsigma\varrho\widehat{q}_k^{(n+1)}}|, \tag{55}$$

assuming $q_k^{(n+1)} \neq 0$ and $q_k^{(n+1)} \neq \varsigma\varrho\widehat{q}_k^{(n+1)}$.

Proof 4. *Considering the HDQDS algorithm in real operations, we have*

$$\begin{aligned} \varsigma\widehat{q}_k^{(n+1)} &= \varsigma\widehat{d}_k^{(n+1)} + \varsigma\widehat{e}_k^{(n)} - \nu_1, \\ \varrho\widehat{q}_k^{(n+1)} &= \varsigma\widehat{q}_k^{(n+1)} + \varsigma\varrho\widehat{q}_k^{(n+1)}, \end{aligned} \tag{56}$$

and using (46), we obtain that

$$\varrho\widetilde{q}_k^{(n+1)} - \widehat{\varsigma}q_k^{(n+1)} = \varsigma\varrho\widehat{d}_k^{(n+1)} + \varsigma\varrho\widehat{e}_k^{(n)}, \tag{57}$$

that derive the bound (50). In addition, we have

$$\begin{aligned} \varsigma\widehat{e}_k^{(n+1)} &= \frac{e_k^{(n)}\varsigma\widehat{q}_{k+1}^{(n)} + \varsigma\widehat{e}_k^{(n)}\widehat{q}_{k+1}^{(n)} - \widehat{e}_k^{(n+1)}\varsigma\widehat{q}_k^{(n+1)} - \nu_3\widehat{e}_k^{(n)} - \nu_4\widehat{q}_k^{(n+1)}}{q_k^{(n+1)}}, \\ \varrho\widehat{e}_k^{(n+1)} &= \varsigma\widehat{e}_k^{(n+1)} + \varsigma\varrho\widehat{e}_k^{(n+1)}, \end{aligned} \tag{58}$$

and considering (47), we get that

$$\varrho\widetilde{e}_k^{(n+1)} - \varsigma\widehat{e}_k^{(n+1)} = \frac{e_k^{(n)}\varsigma\varrho\widehat{q}_{k+1}^{(n)} + \varsigma\varrho\widehat{e}_k^{(n)}q_{k+1}^{(n)} - e_k^{(n+1)}\varsigma\varrho\widehat{q}_k^{(n+1)} - \varsigma\varrho\widehat{e}_k^{(n)}\varsigma\varrho\widehat{q}_{k+1}^{(n)}}{q_k^{(n+1)} - \varsigma\varrho\widehat{q}_k^{(n+1)}}, \tag{59}$$

which can obtain the second bound (51) directly. Identical to the above, we can get the third bound (52). ⊠

Next, we take into account the difference between $\varrho\widetilde{q}_k^{(n+1)}$ and $\varrho\widehat{q}_k^{(n+1)}$, $\varrho\widetilde{e}_k^{(n+1)}$ and $\varrho\widehat{e}_k^{(n+1)}$, and $\varrho\widetilde{d}_{k+1}^{(n+1)}$ and $\varrho\widehat{q}_{k+1}^{(n+1)}$.

Lemma 5. *The difference between $\varrho\widetilde{q}_k^{(n+1)}$ and $\varrho\widehat{q}_k^{(n+1)}$, $\varrho\widetilde{e}_k^{(n+1)}$ and $\varrho\widehat{e}_k^{(n+1)}$, and $\varrho\widetilde{d}_{k+1}^{(n+1)}$ and $\varrho\widehat{q}_{k+1}^{(n+1)}$ are given by*

$$|\varrho\widehat{q}_k^{(n+1)} - \varrho\widetilde{q}_k^{(n+1)}| \leq \gamma_2\gamma_3\big(|d_k^{(n+1)} - \varsigma\varrho\widehat{d}_k^{(n+1)}| + |e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}|\big), \tag{60}$$

$$|\varrho\widehat{e}_k^{(n+1)} - \varrho\widetilde{e}_k^{(n+1)}| \leq \gamma_6\gamma_7\frac{(|e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|)}{|q_k^{(n+1)} - \varsigma\varrho\widehat{q}_k^{(n+1)}|}, \tag{61}$$

and

$$|\varrho\widehat{d}_{k+1}^{(n+1)} - \varrho\widetilde{d}_{k+1}^{(n+1)}| \leq \gamma_8\gamma_9\left(\frac{(|d_k^{(n+1)} - \varsigma\varrho\widehat{d}_k^{(n+1)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|)}{|q_k^{(n+1)} - \varsigma\varrho\widehat{q}_k^{(n+1)}|} + |s^{(n)} - \varsigma\varrho\widehat{s}^{(n)}|\right), \tag{62}$$

where $\varrho\widetilde{q}_k^{(n+1)}$, $\varrho\widetilde{e}_k^{(n+1)}$ and $\varrho\widetilde{d}_{k+1}^{(n+1)}$ are defined in (46), (47) and (48), respectively.

Proof 5. *We first take into account the rounding error of the approximate compensation term* $\varrho\widehat{q}_k^{(n+1)}$ *for* $\widehat{q}_k^{(n+1)}$*, so we have*

$$\varrho\widehat{q}_k^{(n+1)} \approx \varrho\widehat{d}_k^{(n+1)} \oplus \varrho\widehat{e}_k^{(n)} \ominus \nu_1, \tag{63}$$

and using (4) to obtain

$$\varrho\widehat{q}_k^{(n+1)} = \varrho\widehat{d}_k^{(n+1)}(1+\theta_3) + \varrho\widehat{e}_k^{(n)}(1+\theta_2) - \nu_1(1+\theta_1). \tag{64}$$

And from (46), we get

$$|\varrho\widehat{q}_k^{(n+1)} - \varrho\widetilde{q}_k^{(n+1)}| \le \gamma_3(|\varrho\widehat{d}_k^{(n+1)}| + |\varrho\widehat{e}_k^{(n)}| + |\nu_1|). \tag{65}$$

Based on (15), `FastTwoSum`*,* `TwoSum`*,* `TwoProd`*, and* `DivRem` *in Section 3.2, we have*

$$\begin{aligned} &|\nu_1| \le u|\widehat{d}_k^{(n+1)} + \widehat{e}_k^{(n)}| \le u(|\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)}|), \\ &|\nu_1| \le u|(\widehat{d}_k^{(n+1)} + \widehat{e}_k^{(n)})(1+\theta)| \le \gamma_1(|\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)}|). \end{aligned} \tag{66}$$

We update all inputs already using `FastTwoSum` *in the inner loop of each step of the HDQDS algorithm. Thence, according to* `FastTwoSum`*,* `TwoSum`*,* `TwoProd`*, and* `DivRem` *in Section 3.2, we have* $|\varrho\widehat{d}_k^{(n+1)}| \le u|\widehat{d}_k^{(n+1)} - \varrho\widehat{d}_k^{(n+1)}|$ *and* $|\varrho\widehat{e}_k^{(n)}| \le u|\widehat{e}_k^{(n)} - \varrho\widehat{e}_k^{(n)}|$*, where* $\varrho\widehat{d}_k^{(n+1)}$ *and* $\varrho\widehat{e}_k^{(n)}$ *are the updated value. Therefore, from* $u + \gamma_1 \le \gamma_2$*, we get that*

$$\begin{aligned} |\varrho\widehat{q}_k^{(n+1)} - \varrho\widetilde{q}_k^{(n+1)}| \le \gamma_3\Big\{ & u\big(|\widehat{d}_k^{(n+1)} - \varrho\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)} - \varrho\widehat{e}_k^{(n)}|\big) + \gamma_1\big(|\widehat{d}_k^{(n+1)} \\ & - \varrho\widehat{d}_k^{(n+1)}| + |\widehat{e}_k^{(n)} - \varrho\widehat{e}_k^{(n)}|\big)\Big\} \\ \le \gamma_2\gamma_3\big(|d_k^{(n+1)} - \varsigma\varrho\widehat{d}_k^{(n+1)}| + |e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}|\big). \end{aligned} \tag{67}$$

We now discuss the difference between $\varrho\widehat{e}_k^{(n+1)}$ *and* $\varrho\widetilde{e}_k^{(n+1)}$*. According to (58), we obtain*

$$\varrho\widehat{e}_k^{(n+1)} \approx \frac{\widehat{e}_k^{(n)} \otimes \varrho\widehat{q}_{k+1}^{(n)} \oplus \varrho\widehat{e}_k^{(n)} \otimes \widehat{q}_{k+1}^{(n)} \ominus \widehat{e}_k^{(n+1)} \otimes \varrho\widehat{q}_k^{(n+1)} \ominus \nu_3 \otimes \widehat{e}_k^{(n)} \ominus \nu_4 \otimes \widehat{q}_k^{(n+1)}}{\widehat{q}_k^{(n+1)}}, \tag{68}$$

and assuming that all inputs are updated with `FastTwoSum`

$$\widehat{q}_k^{(n+1)} = \widehat{q}_k^{(n+1)} \ominus \varrho\widetilde{q}_k^{(n+1)} = (\widehat{q}_k^{(n+1)} - \varrho\widetilde{q}_k^{(n+1)})(1+\theta_1). \tag{69}$$

From this it can obtain that

$$\begin{aligned} \varrho\widehat{e}_k^{(n+1)} = [&\widehat{e}_k^{(n)}\varrho\widehat{q}_{k+1}^{(n)}(1+\theta_6) + \varrho\widehat{e}_k^{(n)}\widehat{q}_{k+1}^{(n)}(1+\theta_6) - \widehat{e}_k^{(n+1)}\varrho\widehat{q}_k^{(n+1)}(1+\theta_5) \\ &-\nu_3\widehat{e}_k^{(n)}(1+\theta_4) - \nu_4\widehat{q}_k^{(n+1)}(1+\theta_3)]/(\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}). \end{aligned} \tag{70}$$

Thus, from (47), we get

$$|\varrho\widehat{e}_k^{(n+1)} - \varrho\widetilde{e}_k^{(n+1)}| \le \gamma_6(\frac{|\widehat{e}_k^{(n)}||\varrho\widehat{q}_{k+1}^{(n)}| + |\varrho\widehat{e}_k^{(n)}||\widehat{q}_{k+1}^{(n)}| + |\widehat{e}_k^{(n+1)}||\varrho\widehat{q}_k^{(n+1)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|} - \frac{|\nu_3||\widehat{e}_k^{(n)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|} - \frac{|\nu_4||\widehat{q}_k^{(n+1)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|}) + \frac{|\varrho\widehat{e}_k^{(n)}||\varrho\widehat{q}_{k+1}^{(n)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|}. \tag{71}$$

Here, we do not consider the update of the output $\widehat{e}_k^{(n+1)}$*. Therefore, we obtain that*

$$\widehat{e}_k^{(n+1)} = \widehat{e}_k^{(n)} \otimes \widehat{q}_{k+1}^{(n)} \oslash \widehat{q}_k^{(n+1)}(1+\theta_2) \le \widehat{e}_k^{(n)} \times \widehat{q}_{k+1}^{(n)} / \widehat{q}_k^{(n+1)}(1+\gamma_2). \tag{72}$$

Likewise, according to `FastTwoSum`*,* `TwoSum`*,* `TwoProd`*, and* `DivRem` *in Section 3.2, we have* $|\nu_3| \le u|\widehat{q}_{k+1}^{(n)}|$*,* $|\nu_4| \le u|\widehat{e}_{k+1}^{(n)}|$*,* $|\varrho\widehat{e}_k^{(n+1)}| \le u|\widehat{e}_k^{(n+1)} - \varrho\widehat{e}_k^{(n+1)}|$*,* $|\varrho\widehat{q}_{k+1}^{(n)}| \le u|\widehat{q}_{k+1}^{(n)} - \varrho\widehat{q}_{k+1}^{(n)}|$*,* $|\widehat{e}_k^{(n+1)}| \le \frac{1}{1-u}|\widehat{e}_k^{(n+1)} - \varrho\widehat{e}_k^{(n+1)}|$*,* $|\widehat{q}_{k+1}^{(n)}| \le \frac{1}{1-u}|\widehat{q}_{k+1}^{(n)} - \varrho\widehat{q}_{k+1}^{(n)}|$*. From (72), we also obtain* $|\widehat{e}_k^{(n+1)}||\widehat{q}_k^{(n+1)}| \le (1+\gamma_2)|\widehat{e}_k^{(n)}||\widehat{q}_{k+1}^{(n)}|$*.*

Finally, taking into account

$$e_k^{(n+1)} - \varsigma\varrho\widehat{e}_k^{(n+1)} = \widehat{e}_k^{(n+1)} - \varrho\widehat{e}_k^{(n+1)} \tag{73}$$

and (71), with $\gamma_5\gamma_6 + \gamma_1^2 \le \gamma_6\gamma_7$*, we get*

$$\begin{aligned} |\varrho\widehat{e}_k^{(n+1)} - \varrho\widetilde{e}_k^{(n+1)}| &\le \gamma_6\left(\gamma_5 \times \frac{|e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|}\right) \\ &\quad + \gamma_1^2 \frac{|e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|} \\ &\le \gamma_6\gamma_7\left(\frac{|e_k^{(n)} - \varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|}{|\widehat{q}_k^{(n+1)} - \varrho\widehat{q}_k^{(n+1)}|}\right), \end{aligned} \tag{74}$$

which is the second bound.

Similarly, from (48), we get

$$|\varrho\widehat{d}_{k+1}^{(n+1)} - \varrho\widetilde{d}_{k+1}^{(n+1)}| \le \gamma_8\gamma_9\left(\frac{(|d_k^{(n+1)} - \varsigma\varrho\widehat{d}_k^{(n+1)}||q_{k+1}^{(n)} - \varsigma\varrho\widehat{q}_{k+1}^{(n)}|)}{|q_k^{(n+1)} - \varsigma\varrho\widehat{q}_k^{(n+1)}|} + |s^{(n)} - \varsigma\varrho\widehat{s}^{(n)}|\right). \tag{75}$$

⊠

Next, we introduce the rounding error bounds with perturbed input.

Lemma 6. *The rounding error bounds for the HDQDS algorithm's inner loop with perturbed input, are given by*

$$|\varrho\widehat{q}_k^{(n+1)} - \varsigma\widehat{q}_k^{(n+1)}| \le \gamma_3\gamma_4(|d_k^{(n+1)}| + |e_k^{(n)}|) + (1+\gamma_3\gamma_4)(|\varsigma\varrho\widehat{d}_k^{(n+1)}| + \varsigma\varrho\widehat{e}_k^{(n)}|), \tag{76}$$

$$|\varrho\widehat{e}_k^{(n+1)} - \varsigma\widehat{e}_k^{(n+1)}| \leq \gamma_6\gamma_7 c_k^{(n+1)}|e_k^{(n+1)}| + (1+\gamma_6\gamma_7)\omega_k^{(n+1)}, \tag{77}$$

and

$$|\varrho\widehat{d}_{k+1}^{(n+1)} - \varsigma\widehat{d}_{k+1}^{(n+1)}| \leq \gamma_8\gamma_9 c_k^{(n+1)}|d_{k+1}^{(n+1)}| + (1+\gamma_8\gamma_9)\chi_k^{(n+1)}, \tag{78}$$

where $\omega_k^{(n+1)}$, $\chi_k^{(n+1)}$ *and* $c_k^{(n+1)}$ *are defined in (53), (54) and (55), respectively.*

Proof 6. *Taking into account (56) and (58), we have*

$$|\varrho\widehat{q}_k^{(n+1)} - \varsigma\widehat{q}_k^{(n+1)}| \leq |\varrho\widehat{q}_k^{(n+1)} - \varrho\widetilde{q}_k^{(n+1)}| + |\varrho\widetilde{q}_k^{(n+1)} - q_k^{(n+1)}| \tag{79}$$

and from (50) in Lemma 4 and (60) in Lemma 5, we obtain (76).

Then, we get

$$|\varrho\widehat{e}_k^{(n+1)} - \varsigma\widehat{e}_k^{(n+1)}| \leq |\varrho\widehat{e}_k^{(n+1)} - \varrho\widetilde{e}_k^{(n+1)}| + |\varrho\widetilde{e}_k^{(n+1)} - e_k^{(n+1)}| \tag{80}$$

and (81) in Lemma 5, we derive that

$$|\varrho\widehat{e}_k^{(n+1)} - \varrho\widetilde{e}_k^{(n+1)}| \leq c_k^{(n+1)} \times \gamma_6\gamma_7\left(|e_k^{(n+1)}| + \frac{(|e_k^{(n)}||\varsigma\varrho\widehat{q}_{k+1}^{(n)}| + |\varsigma\varrho\widehat{e}_k^{(n)}||q_{k+1}^{(n)}| + |\varsigma\varrho\widehat{q}_{k+1}^{(n)}||\varsigma\varrho\widehat{e}_k^{(n)}|)}{|q_k^{(n+1)}|}\right), \tag{81}$$

with $q_k^{(n+1)} \neq 0$ *and* $\frac{\varsigma\varrho\widehat{q}_k^{(n+1)}}{q_k^{(n+1)}} \neq 1$ *. Hence, considering (62) in Lemma 5, we can obtain the second rounding error bound (77). Finally, we use the same idea to obtain (78).* ⊠

We remark that in Lemma 6 the error bounds for the HDQDS algorithm's inner loop have terms of order $\gamma_6\gamma_7$, and/or terms with compensated approximated terms (terms of type $\varsigma\varrho\square$). So, we have bounds of order greater than one in the roundoff unit, giving more accurate results than the ones in Lemma 3. Therefore, from the error bounds we expect to maintain full precision (that is, 16 digits of precision) up to extremely large problems.

6. Experimental Studies

The following part comprehensively evaluates the performance of our framework. First, we introduce the configuration of the experimental platform (in Section 6.1), and then describe the experimental performance (Sections 6.2 and 6.3).

6.1. Experiment Platform

We have implemented all the algorithms in this work in MATLAB and C. And we test the effectiveness of our framework on Intel, ARM, and AMD processors. All numerical experiments in this paper are carried out under the IEEE-754 standard. Moreover, the data used in the calculations are the floating point numbers. In this paper, we conduct experiments on the Windows system and TianHe supercomputing platform in Changsha Center. The hardware configuration details are shown in Table 2.

Table 2: Configuration Details

Name	Parameters Description
Windows	Core i7-10510U CPU @2.30 GHz
TianHe	Two Intel Xeon Westmere EP @2.50GHz, 12 cores, 48GB memory, CPU peak 140.64 GFlops

6.2. Performance Evaluation of High-Precision DQDS

First, we evaluate the accuracy of the high-precision DQDS algorithm. Comparison of the maximum relative error of q columns calculated by the DQDS, HDQDS, and DD_DQDS algorithms on test matrices of different scales is used to evaluate the precision of the numerical results. The maximum relative error is denoted as

$$\max_{\mathrm{m}}\max_{\mathrm{n}}\{|symq_m^{(n+1)} - q_m^{(n+1)}|/symq_m^{(n+1)}\}. \tag{82}$$

The exact result is obtained by DQDS algorithm with *sym* function in Matlab.

Now, we test the accuracy of the HDQDS algorithm using the shift $s = 0$. As a first test, we use 10 different upper bidiagonal matrices of size $n \times n$ with ranges from $n = 500$ to 600 (middle-scale matrices) [21, 27]. Each element in the diagonals and the upper subdiagonals is a random number from 0 to 1. At the same time, we separately calculate the maximum relative error of all $q_m^{(n+1)}$ obtained by each test matrix, as shown in Fig.3(a). One can find that all the relative error of the HDQDS algorithm is of the order of 10^{-16}, or even 10^{-17}, which is a totally stable behavior like the DD_DQDS algorithm. Note that both algorithms obtain an error of the order of the roundoff unit of the computer. In comparison, the relative error of the DQDS algorithm has apparent fluctuations between 10^{-16} and 10^{-13}. Note that this result shows a rounding error of order $\mathcal{O}(n)$, what is typical of stable algorithms (like the DQDS one) and of the order of just the summation algorithm. The very good results of the tests of the HDQDS algorithm are due to the reduction of the accumulation of rounding errors in the calculation process. In the next experiment, we change the matrix size to use 10 small-scale matrices to demonstrate our proposed algorithm's accuracy. We set the matrices' size range from $n = 5$ to 50. We can observe from Fig.3(b) that our method has higher stability compared to the original DQDS algorithm. We further expand the matrix scale to verify the accuracy of our proposed algorithm, increasing the matrix size from 2000 to 3000. One can see from Fig.3(c) that the maximum relative error of q calculated using the HDQDS algorithm is less than or equal to the working accuracy because EFT minimizes the impact of accuracy loss. The DQDS algorithm is not stable enough when computing the maximum relative error of q for large matrices due to the accumulation of rounding errors.

Next, we set shift s to Johnson shift [18]

$$s^{(n)} = q_m^{(n)} - \sqrt{e_{m-1}^{(n)} q_m^{(n)}} + \frac{1}{4} e_{m-1}^{(n)} > 0. \tag{83}$$

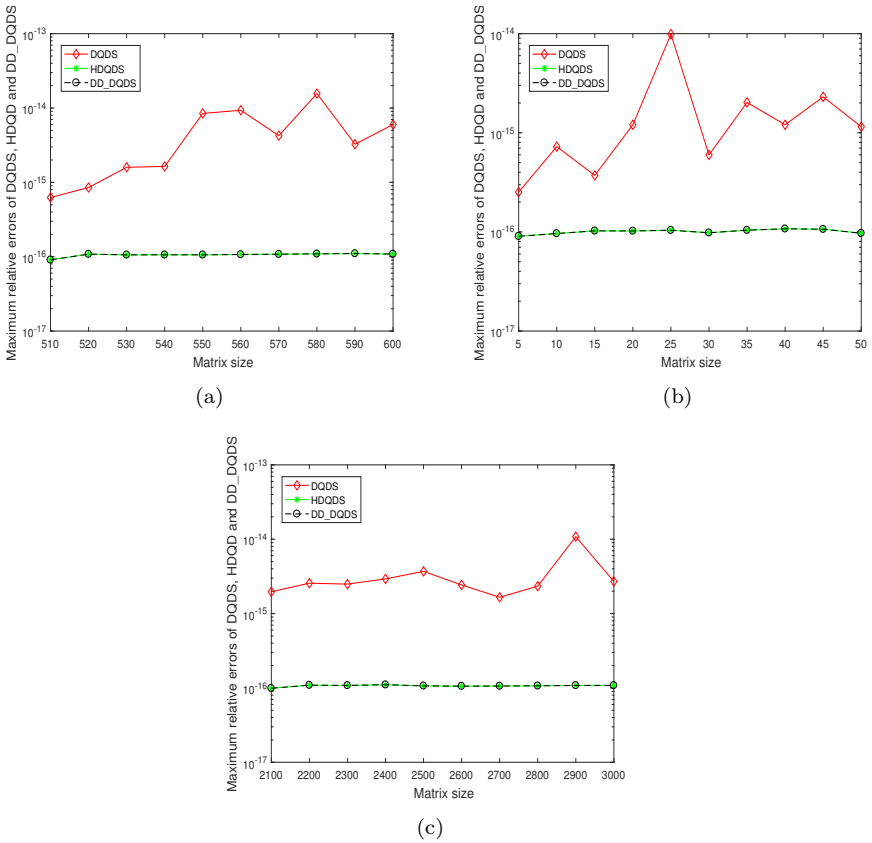

Figure 3: Maximum relative errors on the q columns using the DQDS, HDQDS, and DD_DQDS algorithms without shift. (a) Matrix size n = 500:10:600. (b) Matrix size n = 5:5:50. (c) Matrix size n = 2000:100:3000.

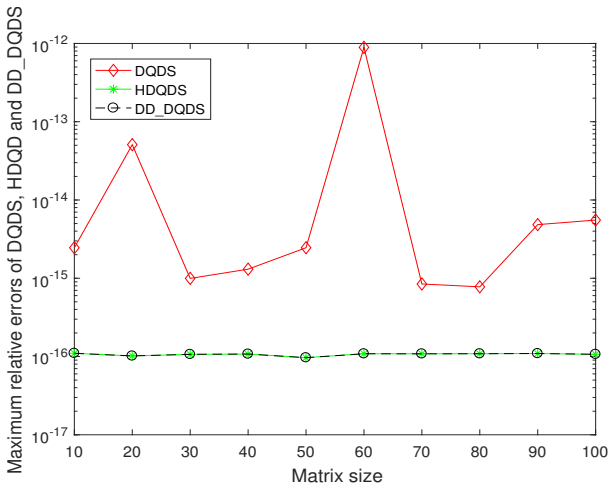

Figure 4: Maximum relative errors on the q columns using the DQDS, HDQDS, and DD_DQDS algorithms using the Johnson shift.

Similarly, we use upper bidiagonal matrices of different scales for testing, each element in the diagonals and the upper subdiagonals is a random number from 0 to 1. The result is shown in Fig.4, presenting a similar performance as before. Therefore, the stability of the HDQDS algorithm is much better than that of the DQDS algorithm, and its accuracy is comparable to the DD_DQDS algorithm.

Besides, we compared the computation time of DQDS, HDQDS and DD_DQDS algorithms on the Tianhe supercomputing platform. We show the statistical experimental results in Fig.5. Each set of experimental results is obtained through multiple tests. One can see from Fig.5 that the running time of our HDQDS is slightly higher than that of the original DQDS. The reason is that HDQDS adds many auxiliary calculation steps to reduce rounding errors. Our target is to reduce the accumulation of rounding errors and improve the accuracy of calculations, so a slight increase in running time is acceptable. One can observe from Fig.3 and Fig.4 that our HDQDS algorithm not only has nearly similar accuracy to DD_DQDS, but its computation time is much less than that of DD_DQDS algorithm. Our HDQDS algorithm almost achieves double-double precision arithmetic operations, so a slight increase in running time is acceptable.

6.3. Performance Evaluation of Mixed-Precision DQDS

In this section, we evaluate the mixed-precision DQDS algorithm's performance by comparing the computation time of MDQDS and DQDS under test matrices of different sizes on the Tianhe supercomputing platform. The test matrices are upper bidiagonal matrices where all the diagonals and the upper subdiagonals are randomly obtained from the interval [0, 1]. Each set of experimental results is obtained through multiple tests.

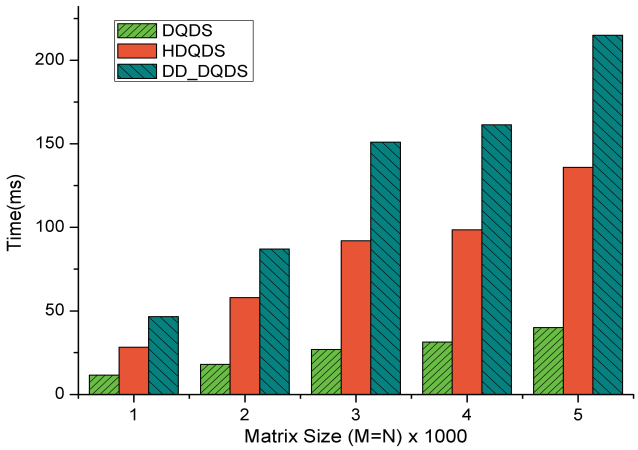

Figure 5: Running Time of DQDS, HDQDS, and DD_DQDS on the CPU.

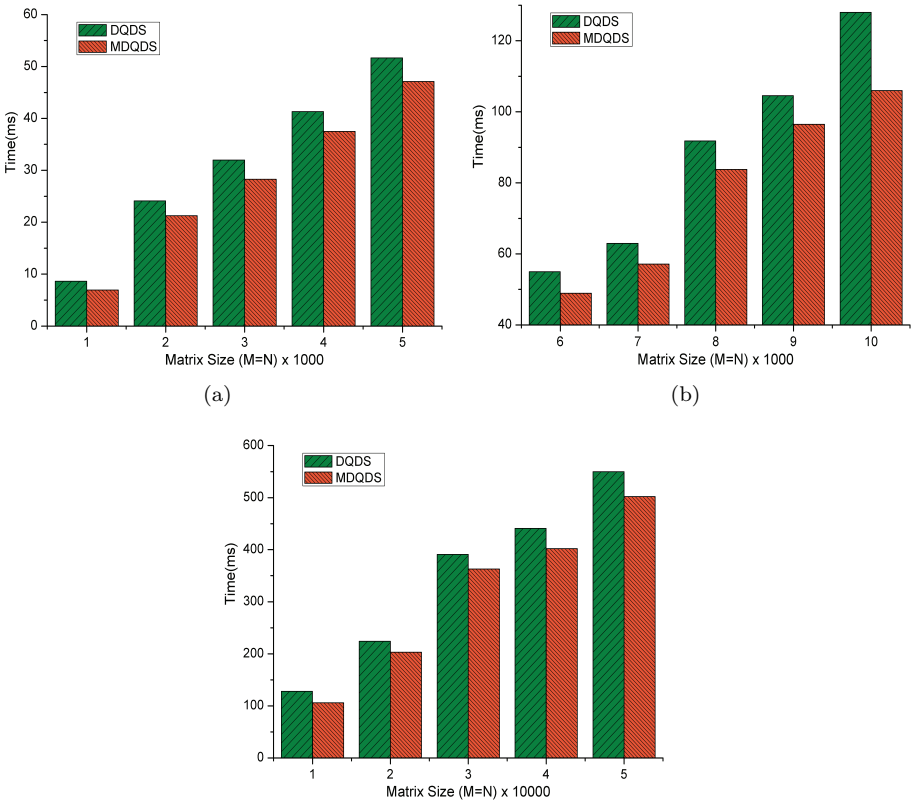

Figure 6: Running Time of DQDS and MDQDS on the CPU without shift. (a) Matrix size n = 1000:1000:5000. (b)Matrix size n = 6000:1000:10000.(c)Matrix size n = 10000:10000:50000.

Firstly, we test the running time of the MDQDS algorithm on the CPU when shift s=0. We test DQDS and MDQDS algorithms using different upper bidiagonal matrices. As shown in Fig.6(a), it can be seen that under the same iterative convergence conditions, the running time of our mixed-precision DQDS algorithm is smaller than that of the DQDS algorithm. The reason is that we use different floating point formats for different operations. The key part uses double-precision, and the other parts use single-precision, which speeds up the calculation while maintaining the same number of iterations. We increase the matrix size from 6000 to 10000, and Fig.6(b) shows that our method has the same number of iterations as the original DQDS algorithm and has less computation time. We further expand the matrix size from 10000 to 50000, and the results are shown in Fig.6(c). With the increase of matrix size, it can be found that the time of both algorithms increases, but the MDQDS algorithm is always lower than the other. Table 3 shows the number of iterations for MDQDS and DQDS. Because we are testing with random matrices, the number of iterations we display is the average of 10 random matrices [21, 27]. This also proves that our MDQDS algorithm can converge faster under the same convergence criterion.

Table 3: Iteration numbers of the MDQDS and DQDS algorithm

Name	5	10	30	50	80	100	300	500	800	1000
MDQDS	5.8	14.4	8.3	6.9	12.9	6.5	5.4	6.6	17.8	7.5
DQDS	5.8	14.4	8.3	6.9	12.9	6.5	5.4	6.6	17.8	7.5

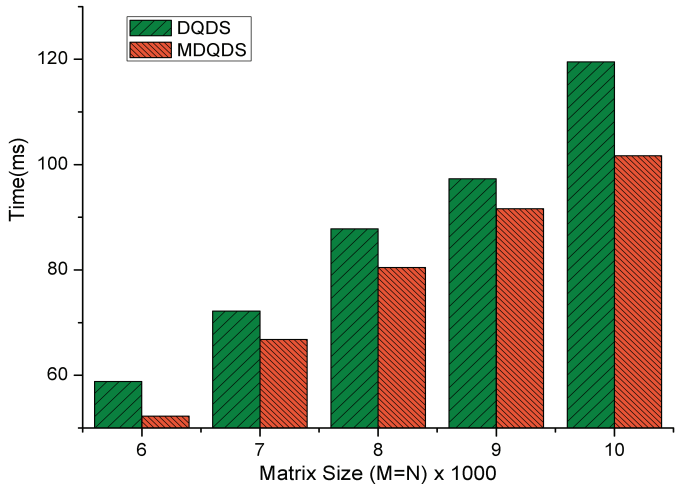

Figure 7: Running Time of DQDS and MDQDS on the CPU using the Johnson shift.

Then we set shift s to Johnson shift, using upper bidiagonal matrices of different scales for testing. Our MDQDS algorithm has the same number of

iterations as the DQDS algorithm under the same iterative convergence conditions, and our MDQDS algorithm spends less computation time shown in Fig.7.

The main use of mixed-precision algorithms is to maintain the convergence properties with the same number of iterations. Users who do not have strict requirements on precision and have requirements on time cost can choose our MDQDS algorithm, which reduces the overhead very well.

7. Conclusions

The DQDS algorithm has the problems of inaccurate results and long calculation time when solving the singular value of the matrix. Based on these drawbacks, this article proposes two new algorithms. One option is to use Error Free Transformations (EFT) technology to decrease rounding errors and ultimately improve accuracy results, which is called the high-precision DQDS algorithm (HDQDS). Additionally, we present the error analysis of the inner loop of the DQDS and HDQDS algorithms. Another algorithm uses the mixed-precision idea to speed up the calculation, where the key steps use double-precision, and the other parts use single-precision, which is called the mixed-precision DQDS algorithm. Furthermore, we propose a precision-adjustable computational framework for solving singular values, named PACF. In our PACF, the same solution algorithm contains three modes: original mode, high-precision mode, and mixed-precision mode. We conduct extensive experiments on supercomputing platforms, and we show that our algorithm outperforms the original algorithm and is credible and efficient. Our framework is easy to operate, which is universal and extensible.

In future work, we prepare to investigate the high-precision and mixed-precision modes of other algorithms for solving singular values of matrices and add them to our framework to further improve performance.

CRediT authorship contribution statement

Chuanying Li: Conduct research, Methodology, Software, Validation, Writing - original draft, Writing - review and editing. Roberto Barrio: Formal analysis, Investigation, Methodology, Writing - review and editing. Xiong Xiao: Validation, Data curation, Writing - review and editing. Peibing Du: Conceptualization, Methodology, Writing - review and editing. Hao Jiang: Formal analysis, Validation, Funding acquisition. Zhe Quan: Supervision, Resources, Funding acquisition. Kenli Li: Supervision, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by the 173 program of China (2020-JCJQ-ZD-029), National Natural Science Foundation of China(No. 61907034), the National Key Research and Development Program of China(2020YFA0709803), the Science Challenge Project of China(TZ2016002), the European Regional Development Fund and Spanish Research projects (PGC2018-096026-B-I00 and PID2021-122961NB-I00) and the European Regional Development Fund and Diputación General de Aragón (E24-17R).

References

[1] L. Xu, S. Chatterton, P. Pennacchi, Rolling element bearing diagnosis based on singular value decomposition and composite squared envelope spectrum, Mechanical Systems and Signal Processing 148 (2021) 107174. doi:https://doi.org/10.1016/j.ymssp.2020.107174.

[2] Y.-T. Shih, C.-S. Chien, C.-Y. Chuang, An adaptive parameterized block-based singular value decomposition for image de-noising and compression, Applied Mathematics and Computation 218 (2012) 10370–10385. doi:https://doi.org/10.1016/j.amc.2012.03.070.

[3] W. G. Hoover, C. G. Hoover, F. Grond, Phase-space growth rates, local Lyapunov spectra, and symmetry breaking for time-reversible dissipative oscillators, Communications in Nonlinear Science and Numerical Simulation 13 (6) (2008) 1180–1193. doi:https://doi.org/10.1016/j.cnsns.2006.10.008.

[4] Y. Hu, H. Chen, L. Lu, H. Zhang, K. J. Marfurt, An adaptive noise-free method of seismic resolution enhancement based on extrapolated multiresolution singular value decomposition, IEEE Transactions on Geoscience and Remote Sensing 58 (7) (2020) 4958–4966. doi:https://doi.org/10.1109/TGRS.2020.2970732.

[5] N. Muller, L. Magaia, B. M. Herbst, Singular value decomposition, eigenfaces, and 3D reconstructions, SIAM Review 46 (3) (2004) 518–545. doi:10.1137/S0036144501387517.

[6] N. Le Bihan, S. J. Sangwine, Jacobi method for quaternion matrix singular value decomposition, Applied Mathematics and Computation 187 (2) (2007) 1265–1271. doi:https://doi.org/10.1016/j.amc.2006.09.055.

[7] H.-S. Ahn, Y. Chen, Exact maximum singular value calculation of an interval matrix, IEEE transactions on automatic control 52 (3) (2007) 510–514. doi:https://doi.org/10.1109/TAC.2006.890475.

[8] P. Du, R. Barrio, H. Jiang, L. Cheng, Accurate quotient-difference algorithm: error analysis, improvements and applications, Applied Mathematics and Computation 309 (2017) 245–271. doi:https://doi.org/10.1016/j.amc.2017.04.004.

[9] K. Ozaki, An error-free transformation for matrix multiplication with reproducible algorithms and divide and conquer methods, in: Journal of Physics: Conference Series, Vol. 1490, IOP Publishing, 2020, p. 012062. doi:https://doi.org/10.1088/1742-6596/1490/1/012062.

[10] T. Ogita, S. M. Rump, S. Oishi, Accurate sum and dot product, SIAM Journal on Scientific Computing 26 (6) (2005) 1955–1988. doi:https://doi.org/10.1137/030601818.

[11] H. Jiang, S. Graillat, C. Hu, S. Li, X. Liao, L. Cheng, F. Su, Accurate evaluation of the k-th derivative of a polynomial and its application, Journal of Computational and Applied Mathematics 243 (2013) 28–47. doi:https://doi.org/10.1016/j.cam.2012.11.008.

[12] S. Graillat, F. Jézéquel, R. Picot, Numerical validation of compensated algorithms with stochastic arithmetic, Applied Mathematics and Computation 329 (2018) 339–363. doi:https://doi.org/10.1016/j.amc.2018.02.004.

[13] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, A. Fox, M. Gates, N. J. Higham, X. S. Li, et al., A survey of numerical linear algebra methods utilizing mixed-precision arithmetic, The International Journal of High Performance Computing Applications 35 (4) (2021) 344–369. doi:https://doi.org/10.1177/10943420211003313.

[14] N. Lindquist, P. Luszczek, J. Dongarra, Accelerating restarted GMRES with mixed precision arithmetic, IEEE Transactions on Parallel and Distributed Systems 33 (2022) 1027–1037. doi:https://doi.org/10.1109/TPDS.2021.3090757.

[15] J. Sun, G. D. Peterson, O. O. Storaasli, High-performance mixed-precision linear solver for FPGAs, IEEE Transactions on Computers 57 (12) (2008) 1614–1623. doi:https://doi.org/10.1109/TC.2008.89.

[16] K. V. Fernando, B. N. Parlett, Accurate singular values and differential qd algorithms, Numerische Mathematik 67 (2) (1994) 191–229. doi:https://doi.org/10.1007/s002110050024.

[17] B. N. Parlett, O. A. Marques, An implementation of the dqds algorithm (positive case), Linear Algebra and its Applications 309 (1-3) (2000) 217–259. doi:https://doi.org/10.1016/S0024-3795(00)00010-0.

[18] K. Aishima, T. Matsuo, K. Murota, M. Sugihara, On convergence of the DQDS algorithm for singular value computation, SIAM Journal on Matrix

Analysis and Applications 30 (2) (2008) 522–537. doi:https://doi.org/10.1137/060678762.

[19] K. Aishima, T. Matsuo, K. Murota, Rigorous proof of cubic convergence for the dqds algorithm for singular values, Japan Journal of Industrial and Applied Mathematics 25 (1) (2008) 65–81. doi:https://doi.org/10.1007/BF03167513.

[20] K. Aishima, T. Matsuo, K. Murota, M. Sugihara, A survey on convergence theorems of the dqds algorithm for computing singular values, J. Math-for-Ind 2 (2010) 1–11.

[21] K. Aishima, T. Matsuo, K. Murota, M. Sugihara, A shift strategy for superquadratic convergence in the dqds algorithm for singular values, Journal of Computational Applied Mathematics 257 (2014) 132–143. doi:https://doi.org/10.1016/j.cam.2013.08.021.

[22] K. Aishima, T. Matsuo, K. Murota, M. Sugihara, Superquadratic convergence of DLASQ for computing matrix singular values, Journal of computational and applied mathematics 234 (4) (2010) 1179–1187. doi:https://doi.org/10.1016/j.cam.2009.07.021.

[23] S. Li, M. Gu, B. N. Parlett, An improved dqds algorithm, SIAM Journal on Scientific Computing 36 (3) (2014) C290–C308. doi:https://doi.org/10.1137/120881087.

[24] Y. Nakatsukasa, K. Aishima, I. Yamazaki, dqds with aggressive early deflation, SIAM Journal on Matrix Analysis and Applications 33 (1) (2012) 22–51. doi:https://doi.org/10.1137/110821330.

[25] S. Araki, H. Tanaka, M. Takata, K. Kimura, Y. Nakamura, Fast computation method of column space by using the DQDS method and the OQDS method, in: Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), 2018, pp. 333–339.

[26] C. Ferreira, B. Parlett, Real dqds for the nonsymmetric tridiagonal eigenvalue problem, arXiv preprint arXiv:1201.5065.

[27] T. Yamashita, K. Kimura, M. Takata, Y. Nakamura, An application of the Kato-Temple inequality on matrix eigenvalues to the dqds algorithm for singular values, JSIAM Letters 5 (2013) 21–24. doi:https://doi.org/10.14495/jsiaml.5.21.

[28] C. Li, X. Xiao, P. Du, H. Jiang, R. Barrio, Z. Quan, K. Li, A high-precision DQDS algorithm, in: 2021 IEEE 23st International Conference on High Performance Computing and Communications; IEEE 19th International Conference on Smart City; IEEE 7th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), IEEE, 2021.

[29] R. Barrio, P. Du, H. Jiang, S. Serrano, ORTHOPOLY: A library for accurate evaluation of series of classical orthogonal polynomials and their derivatives, Computer Physics Communications 231 (2018) 146–162. doi:https://doi.org/10.1016/j.cpc.2018.05.004.

[30] M. Baboulin, A. Buttari, J. Dongarra, J. Kurzak, J. Langou, J. Langou, P. Luszczek, S. Tomov, Accelerating scientific computations with mixed precision algorithms, Computer Physics Communications 180 (12) (2009) 2526–2533. doi:https://doi.org/10.1016/j.cpc.2008.11.005.

[31] A. Abdelfattah, H. Anzt, E. G. Boman, E. Carson, T. Cojean, J. Dongarra, M. Gates, T. Grützmacher, N. J. Higham, S. Li, et al., A survey of numerical methods utilizing mixed precision arithmetic, arXiv preprint arXiv:2007.06674.

[32] X. S. Li, J. W. Demmel, D. H. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Y. Kang, A. Kapur, M. C. Martin, et al., Design, implementation and testing of extended and mixed precision BLAS, ACM Transactions on Mathematical Software (TOMS) 28 (2) (2002) 152–205. doi:https://doi.org/10.1145/567806.567808.

[33] I. Yamazaki, S. Tomov, J. Dongarra, Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs, SIAM Journal on Scientific Computing 37 (3) (2015) C307–C330. doi:https://doi.org/10.1137/14M0973773.

[34] N. J. Higham, X. Liu, A multiprecision derivative-free Schur–Parlett algorithm for computing matrix functions, SIAM Journal on Matrix Analysis and Applications 42 (3) (2021) 1401–1422. doi:https://doi.org/10.1137/20M1365326.

[35] P. Sun, Y. Wen, R. Han, W. Feng, S. Yan, Gradientflow: Optimizing network performance for large-scale distributed dnn training, IEEE Transactions on Big Data 8 (2022) 495–507. doi:https://doi.org/10.1109/TBDATA.2019.2957478.

[36] M. H. Gutknecht, B. N. Parlett, From qd to LR, or, how were the qd and LR algorithms discovered?, IMA journal of numerical analysis 31 (3) (2011) 741–754. doi:https://doi.org/10.1093/imanum/drq003.

[37] L. M. Yang, A. Fox, G. Sanders, Rounding error analysis of mixed precision block Householder QR algorithms, SIAM Journal on Scientific Computing 43 (3) (2021) A1723–A1753. doi:https://doi.org/10.1137/19M1296367.

[38] N. J. Higham, Accuracy and stability of numerical algorithms, second ed., SIAM, 2002.

[39] T. J. Dekker, A floating-point technique for extending the available precision, Numerische Mathematik 18 (3) (1971) 224–242. doi:https://doi.org/10.1007/BF01397083.

[40] D. E. Knuth, Art of computer programming, volume 2: Seminumerical algorithms, third ed., Addison-Wesley Professional, 2014.

[41] J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, S. Torres, The Fused Multiply-Add Instruction, Birkhäuser Boston, Boston, 2010, pp. 151–179. doi:10.1007/978-0-8176-4705-6_5.

[42] N. Louvet, Compensated algorithms in floating point arithmetic: accuracy, validation, performances, Ph.D. thesis, Université de Perpignan Via Domitia, 2007.

[43] M. Pichat, J. Vignes, Ingénierie du contrôle de la précision des calculs sur ordinateur, Editions Technip, 1993.

[44] S. Graillat, P. Langlois, N. Louvet, Algorithms for accurate, validated and fast polynomial evaluation, Japan Journal of Industrial and Applied Mathematics 26 (2) (2009) 191–214. doi:https://doi.org/10.1007/BF03186531.

[45] D. H. Bailey, Qd library in high-precision software directory, https://www.davidhbailey.com/dhbsoftware/.

[46] Y. Hida, X. S. Li, D. H. Bailey, Algorithms for quad-double precision floating point arithmetic, in: Proceedings 15th IEEE Symposium on Computer Arithmetic. ARITH-15 2001, IEEE, 2001, pp. 155–162. doi:https://doi.org/10.1109/ARITH.2001.930115.