Applied Mathematics and Computation 273 (2016) 1160–1178

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Accurate, validated and fast evaluation of elementary symmetric functions and its application

Hao Jiang[a,1,*], Stef Graillat[b], Roberto Barrio[c,2], Canqun Yang[d,3]

[a] School of Computer Science, National University of Defense Technology, Changsha 410073, China
[b] PEQUAN, LIP6, Université Pierre et Marie Curie, Paris 75025, France
[c] Dpto. de Matemática Aplicada and IUMA, Universidad de Zaragoza, Zaragoza E-50009, Spain
[d] Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha 410073, China

ARTICLE INFO

Keywords:
Elementary symmetric functions
Floating-point arithmetic
Roundoff error
Error-free transformation
Compensated algorithm
Accurate algorithm

ABSTRACT

This paper is concerned with the fast, accurate and validated evaluation of elementary symmetric functions in floating-point arithmetic. We present two new compensated algorithms, with real and complex floating-point inputs respectively, by applying error-free transformations to improve the accuracy of the so-called summation algorithm that is used, by example, in the MATLAB's `poly` function. We derive forward roundoff error bounds and running error bounds for our new algorithms. The roundoff error bounds imply that the computed results are as accurate as if computed with twice the working precision and then rounded to the current working precision. The running error analysis provides a shaper bound along with the result, without increasing significantly the computational cost. Numerical experiments illustrate that our algorithms run much faster than the algorithms using the classic double–double library while sharing similar error estimates. Such algorithms can be widely applicable for example to compute characteristic polynomials from eigenvalues or polynomial's coefficients from zeros. Some simple applications are presented to show that the proposed algorithms compute the coefficients of the characteristic polynomials of some real and complex matrices to high relative accuracy.

1. Introduction

kth Elementary Symmetric Function (ESF) associated with a vector of n numbers $X = (x_1, \ldots, x_n)$ is defined as

$$S_k^{(n)}(X) = \sum_{1\le\pi_1<\ldots<\pi_k\le n} x_{\pi_1}x_{\pi_2}\ldots x_{\pi_k},\ 1 \le k \le n, \tag{1}$$

* Corresponding author. Tel.: +86 73184574650.
E-mail addresses: haojiang@nudt.edu.cn, jhnudt@163.com (H. Jiang), stef.graillat@upmc.fr (S. Graillat), rbarrio@unizar.es (R. Barrio), canqun@nudt.edu.cn (C. Yang).
1 Partially supported by National Natural Science Foundation of China (No. 61402495, No. 61303189).
2 Partially supported by the Spanish Research project MTM2012-31883 and by Gobierno de Aragon (group E48) and Fondo Social Europeo.
3 Partially supported by National High Technology Research and Development Program of China (No. 2012AA01A301).

http://dx.doi.org/10.1016/j.amc.2015.08.134
0096-3003/© 2015 Elsevier Inc. All rights reserved.

which consists of $\binom{n}{k}$ summands. For $k = 0$, $S_0^{(n)}(X) = 1$. Throughout this paper, we assume that the inputs $X = (x_1, \ldots, x_n)$ are real or complex floating-point numbers.

The classic and widely-used method to compute the elementary symmetric function (1) is the so-called summation algorithm [1], which is essentially the algorithm used by MATLAB's `poly` function. The error analysis of this algorithm has been considered in [2], and the result implies the algorithm is stable. However, as mentioned in [2] "due to cancellation from subtraction", for some too ill-conditioned problems, the computed result by the summation algorithm in floating-point arithmetic may be still little accurate. Then a higher accurate algorithm is required.

In this paper, motivated by the papers [3–10], we propose two fast and accurate compensated algorithms with real and complex inputs respectively, by introducing error-free transformation (EFT) to the traditional summation algorithm. We focus mainly on the case $2 \le k \le n-1$. For $k = 1$, the problem simplifies to computing a sum of floating-point numbers, and for $k = n$, to computing a floating-point product. The corresponding compensated algorithms for these two cases with real floating-point numbers can be found in [7] and [3], respectively.

As an application, the ESFs appear when expanding a linear factorization of a polynomial

$$\prod_{i=1}^{n}(x - x_i) = \sum_{i=0}^{n} c_i x^{n-i} = \sum_{i=0}^{n}(-1)^i S_i^{(n)}(X) x^{n-i}. \tag{2}$$

With the Summation algorithm, one can evaluate polynomial's coefficients $\{c_i\}_{i=1}^n$ from its zeros $\{x_i\}_{i=1}^n$ ($c_0 = 1$), and specially can compute characteristic polynomials from eigenvalues (see [2,11,12]). Our algorithms can be used to enhance the accuracy for some ill-conditioned polynomials' coefficients evaluation.

The computation of ESFs is also an important part of conditional maximum likelihood estimation (CMLE) of item parameters under the Rasch model in psychological measurement [13]. It is promising that our algorithm, improving the numerical accuracy, can allow much more items to be calibrated.

The rest of the paper is organized as follows. In Section 2, we introduce some basic notations and results about floating-point arithmetic, error-free transformations and the condition number of the problem. After that we recall the summation algorithm, denoted by `SumESF`. In Section 3, we first focus on the case of real floating-point inputs. We propose a new compensated summation algorithm, denoted by `CompSumESF`, together with an error bound. To obtain a sharper error bound, the corresponding running error analysis is performed. We also present an accurate algorithm using the double–double library, denoted by `DDSumESF`, which is used to compare with our compensated algorithm. In Section 4, we extend the results in Section 3 to the case of complex floating-point inputs. We use the complex error-free transformations to obtain the compensated summation algorithm in complex floating-point arithmetic and present its error analysis. In Section 5, numerical experiments illustrate the accuracy and efficiency of our compensated algorithms. Finally, concluding remarks and future work are left for Section 6.

This paper is an extended version of [14] that only dealt with real floating-point inputs. Here the novelty is that we also consider the accurate and fast evaluation of ESFs in complex floating-point arithmetic. We also performed more performance tests on different environments. Moreover, we present its well application of computing the characteristic polynomial from matrix.

2. Notations and preliminaries

2.1. Floating-point arithmetic

In this paper we assume all the floating-point computations are performed in double precision (binary64 in IEEE-754 2008 standard), with "rounding to the nearest" mode and no underflow nor overflow occurring. We also assume that the computations in floating-point arithmetic follow the model

$$a \text{ op } b = fl(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2), \tag{3}$$

where op $\in \{\oplus, \ominus, \otimes\}$, $\circ \in \{+, -, \times\}$ and $|\varepsilon_1|, |\varepsilon_2| \le u$ and $fl(\cdot)$ denotes the result of a floating-point computation, where all operations inside parentheses are done in floating-point arithmetic. The symbol u is the round-off unit, for IEEE 754 double precision, $u = 2^{-53}$, and 'fl' represents the floating-point computation, e.g. $a \oplus b = fl(a + b)$. We denote the computed result of $a \in \mathbb{R}$ in floating-point arithmetic by $\hat{a}$ or fl(a) and the set of all floating-point numbers by $\mathbb{F}$. Following [15], we also use the following classic properties in error analysis (we always assume that $nu < 1$).

1. if $|\delta_i| \le u$, $\rho_i = \pm 1$, then $\prod_{i=1}^n (1 + \delta_i)^{\rho_i} = 1 + \theta_n$,
2. $1 + \theta_n = <n>$ and $|\theta_n| \le \gamma_n := nu/(1 - nu)$,
3. $(1 + \theta_k)(1 + \theta_j) = (1 + \theta_{k+j})$, $<k><j> = <k + j>$,
4. $\gamma_k + \gamma_j + \gamma_k\gamma_j \le \gamma_{k+j}$ and $\gamma_k < \gamma_{k+1}$.

To derive the running error bound, we need the next relations obtained from [4] and [18].

$$\gamma_k \le (1 + u)\widehat{\gamma}_k, \; (1 + u)^n |x| \le fl\Big(\frac{|x|}{1 - (n+1)u}\Big). \tag{4}$$

Table 1
Error-free transformations of addition and product.

Algorithm	Flops	Ref.
$[x, y] = \texttt{TwoSum}(a, b)$	6	[16]
$[x, y] = \texttt{Split}(a)$	4	[17]
$[x, y] = \texttt{TwoProd}(a, b)$	17	[17]
$[x, y] = \texttt{FastTwoSum}(a, b)$	3	[17]
$[x, y] = \texttt{TwoProdFMA}(a, b)$	2	[7]
$[x, y] = \texttt{TwoSumCplx}(a, b)$	12	[5]
$[p, e, f, g] = \texttt{TwoProdCplx}(a, b)$	80	[5]
$[p, e, f, g] = \texttt{TwoProdCplxSingleSplitting}(a, b)$	64	[5]
$[p, e, f, g] = \texttt{TwoProdFMACplx}(a, b)$	20	[5]

To carry out error analysis in complex arithmetic, we use the following model from [15]

$$\begin{aligned} fl(a \pm b) &= (a \pm b)(1 + \delta_1), \ |\delta_1| < u, \\ fl(ab) &= ab(1 + \delta_2), \ |\delta_2| < \sqrt{2}\gamma_2, \end{aligned} \tag{5}$$

with $a, b, \delta_1, \delta_2 \in \mathbb{F} + i\mathbb{F}$, where the absolute value of a complex number $x = x_r + ix_i$ is defined by $|x| = \sqrt{x_r^2 + x_i^2}$. For $a, b \in \mathbb{F} + i\mathbb{F}$, there are properties

$$|a + b| \le |a| + |b|, \ |a \cdot b| = |a| \cdot |b|. \tag{6}$$

We also use the notation $\widetilde{\gamma}_n$ from [5] with $u_c = \sqrt{2}\gamma_2$

$$\widetilde{\gamma}_n = \frac{nu_c}{1 - nu_c} = \frac{n\sqrt{2}\gamma_2}{1 - n\sqrt{2}\gamma_2}. \tag{7}$$

2.2. Error-free transformations

For a pair of floating-point numbers $a, b \in \mathbb{F}$, when no underflow nor overflow occur, there exists a floating-point number y satisfying $a \circ b = x + y$, where $x = \mathrm{fl}(a \circ b)$ and $\circ \in \{+, -, \times\}$. The transformation $(a, b) \longrightarrow (x, y)$ is regarded as an error-free transformation. The error-free transformation algorithms of the addition and product of two floating-point numbers used later in this paper are mainly `TwoSum` and `TwoProd` algorithms, respectively. If the relative sizes of the operands of the addition are known *a priori*, and the comparison can be avoided, then `FastTwoSum` may be faster than `TwoSum`. On some computers where Fused-Multiply-and-Add (FMA) operator is available, `TwoProd` can be implemented more efficiently by being rewritten as `TwoProdFMA`. We can see the details of the four algorithms above in the references presented in Table 1. Here, algorithm `Split`, which can split a floating-point number into two parts, is used in `TwoProd`.

The following theorem summarizes the properties of algorithm `TwoSum` and `TwoProd`.

Theorem 1 (Ogita et al. [7]). *For $a, b, x, y \in \mathbb{F}$, $[x, y] = \texttt{TwoSum}(a, b)$ verifies*

$$x + y = a + b, \quad x = \mathrm{fl}(a + b), \quad y \le u|x|, \quad y \le u|a + b|;$$

and for $a, b, x, y \in \mathbb{F}$, $[x, y] = \texttt{TwoProd}(a, b)$ verifies

$$x + y = a \times b, \quad x = \mathrm{fl}(a \times b), \quad y \le u|x|, \quad y \le u|a \times b|.$$

The complex error-free transformations and corresponding error analysis are presented in [19] and [5] with the theorem below.

Theorem 2 (Graillat et al. [5]). *For $a, b, x, y \in \mathbb{F} + i\mathbb{F}$, $[x, y] = \texttt{TwoSumCplx}(a, b)$ verifies*

$$x + y = a + b, \quad x = \mathrm{fl}(a + b), \quad y \le u|x|, \quad y \le u|a + b|.$$

Meanwhile for $a, b, p, e, f, g \in \mathbb{F} + i\mathbb{F}$, $[p, e, f, g] = \texttt{TwoProdCplx}(a, b)$ verifies

$$p + e + f + g = a \times b, \quad p = \mathrm{fl}(a \times b), \quad |e + f + g| \le \sqrt{2}\gamma_2|a \times b|.$$

2.3. Condition number

Condition numbers measure the sensitivity of the solution of a problem to perturbation in the data. To perform error analysis, we define this condition number of the kth ESF evaluation (1) as

$$\texttt{cond}(S_k^{(n)}(X)) = \lim_{\varepsilon \to 0} \sup \left\{ \frac{|S_k^{(n)}(X + \triangle X) - S_k^{(n)}(X)|}{\varepsilon |S_k^{(n)}(X)|} : |\triangle X| < \varepsilon |X| \right\},$$

Algorithm 1: Summation Algorithm

```
Input: $X = (x_1, \dots, x_n)$ and $k$
Output: k-th ESF $S_k^{(n)}(X) = S_k^{(n)}$
function $S_k^{(n)}$=SumESF$(X, k)$
$S_0^{(i)} = 1,\ 1 \le i \le n-1;\ S_j^{(i)} = 0,\ j > i;\ S_1^{(1)} = x_1;$
for $i = 2 : n$
    for $j = \max\{1, i+k-n\} : \min\{i, k\}$
        $S_j^{(i)} = S_j^{(i-1)} + x_i S_{j-1}^{(i-1)};$
    end
end
```

where absolute value and comparison are to be understood componentwise. A direct calculation yields:

$$\texttt{cond}\big(S_k^{(n)}(X)\big) = \frac{kS_k^{(n)}(|X|)}{|S_k^{(n)}(X)|}. \tag{8}$$

In particular, $\texttt{cond}(S_n^{(n)}(X)) = \texttt{cond}(\prod_{i=1}^n x_i) = n$ and $\texttt{cond}(S_1^{(n)}(X)) = \texttt{cond}(\sum_{i=1}^n x_i) = \frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|}$.

It is direct that in complex arithmetic case the definition of condition number is the same.

2.4. Classic algorithm

The summation algorithm, represented by Algorithm 1 below, computes the elementary symmetric functions recursively, which is the same as the one in [2], except that it only computes the kth ESF rather than all of ESFs.

Here, it is obvious that $S_j^{(i)}$ is an abbreviation of

$$S_j^{(i)} = S_j^{(i)}(x_1, \dots, x_i) = \sum_{1 \le \pi_1 < \dots < \pi_j \le i} x_{\pi_1} x_{\pi_2} \dots x_{\pi_j}.$$

If we substitute $j = 1 : i$ for $j = \max\{1, i+k-n\} : \min\{i, k\}$, we can compute all ESFs simultaneously. For the simplification of the error analysis, we only consider the computation of the kth ESF. However, in practical calculation such as computing characteristic polynomial from eigenvalues, this substitution is often required.

The following theorem gives the roundoff error bounds for Algorithm 1.

Theorem 3. *If $X = (x_1, \dots, x_n)$ is a vector of floating-point numbers, the computed kth elementary symmetric function $\widehat{S}_k^{(n)} = \widehat{S}_k^{(n)}(X)$ by Algorithm 1 in floating-point arithmetic verifies*

$$\begin{array}{lcl} \left|\frac{\widehat{S}_k^{(n)} - S_k^{(n)}}{S_k^{(n)}}\right| & \le & \frac{1}{k}\gamma_{2(n-1)}\texttt{cond}(S_k^{(n)}),\ 2 \le k \le n-1, \\ \left|\frac{\widehat{S}_1^{(n)} - S_1^{(n)}}{S_1^{(n)}}\right| & \le & \gamma_{n-1}\texttt{cond}(S_1^{(n)}) = \gamma_{n-1}\frac{\sum_{i=1}^n |x_i|}{|\sum_{i=1}^n x_i|},\ k = 1, \\ \left|\frac{\widehat{S}_n^{(n)} - S_n^{(n)}}{S_n^{(n)}}\right| & \le & \frac{1}{n}\gamma_{n-1}\texttt{cond}(S_n^{(n)}) = \gamma_{n-1},\ k = n. \end{array}$$

Proof. For the cases of $k = 1$ and $k = n$, the results directly come from Lemma 8.4 and Lemma 3.1 of [15], respectively. For the case of $2 \le k \le n-1$, there are two methods to obtain the error bound. One method is the induction shown in [2]. However we deem that the error bound of $\theta_t^{(i_1 \dots i_k)}$ in Theorem 4.3 of [2] should be $\gamma_{2(n-1)}$. Hence, we make a small improvement by substituting $\gamma_{2(n-1)}$ for γ_{2n}. The other one is using data dependency graph just like that in [20], [21] and [22]. It is easy to obtain the following equation

$$|\widehat{S}_j^{(i)} - S_j^{(i)}| \le \gamma_{2(i-1)} S_j^{(i)}(|x_1|, \dots, |x_i|),\ 1 \le j \le i \le n. \tag{9}$$

Let $j = k$ and $i = n$, we will obtain the expected result. Finally by definition of the condition number (8), we can obtain the relative error bound of Algorithm 1. □

Here, we briefly introduce the data dependency graph to show its effectiveness in the error analysis. In Fig. 1, there exists the following relation:

$$S(i, j) = V\_arr(i, j) \times S(i-1, j) + D\_arr(i, j) \times S(i-1, j-1) + C(i, j), \tag{10}$$

where $S(i, j)$ is the computed values, the vertical arrows $V\_arr(i, j)$ and the diagonal arrows $D\_arr(i, j)$ are coefficients, and $C(i; j)$ is a constant. Taking how to get the error bound of $\theta_t^{(i_1 \dots i_k)}$ in the proof of Theorem 3 as an example. Let $S(i, j) = \widehat{S}_j^{(i)}$,

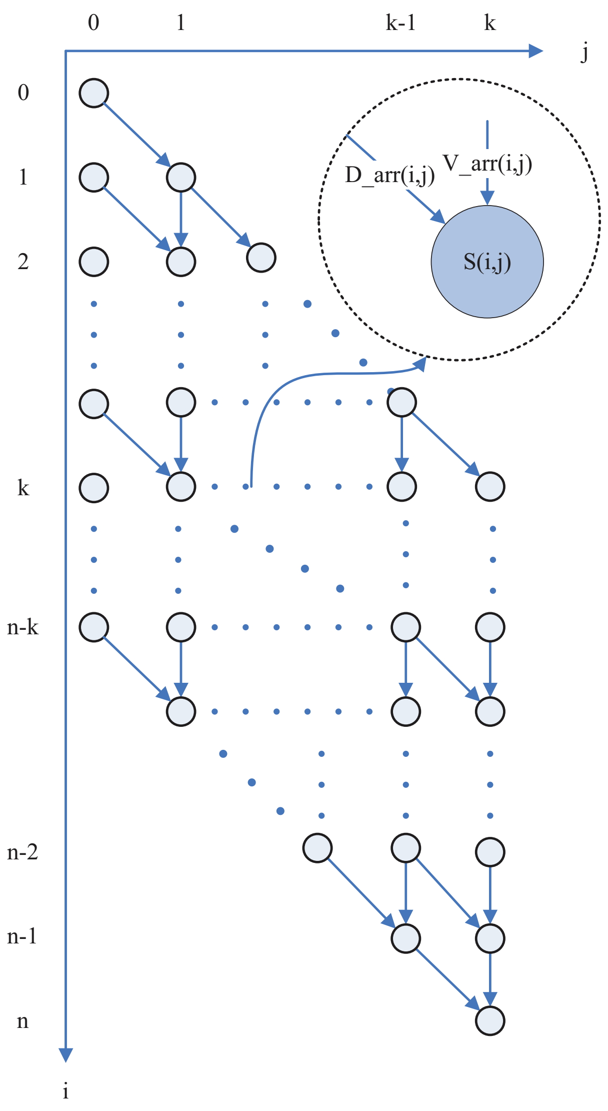

Fig. 1. Data dependency graph for summation algorithm and Compensated summation algorithm.

$V\_arr(i,j) = <1>$, and $C(i,j) = 0$ for $1 \le j \le i \le n$; $D\_arr(i,j) = x_i<2>$ for $2 \le j \le i-1$, $3 \le i \le n$; and $D\_arr(i,j) = x_i<1>$ for $1 \le i = j \le k$ or $j = 1,\ i = 1 : n-k+1$. Based on the data dependency graph, it is easy to obtain that the most perturbation of $\widehat{S}_j^{(i)}$ is $<2(j-1)+(i-j)>$, the bound of which is $\gamma_{2(i-1)}$ due to $j \le i$.

In the case of complex floating-point inputs, classic SumESF needs no modification but to perform in complex arithmetic. The error bounds for complex values are derived in [23]. To distinguish these two cases, we denote SumESF in complex floating-point case by SumESFCplx. Substituting u_c for u and $\widetilde{\gamma}$ for γ in (7), we can get its corresponding forward error bounds. Actually, for $k = 1$, according to the addition model in (5), we deem that substitution is not necessary.

3. Accurate ESF evaluation

In this section, we focus on the real floating-point arithmetic case. We present a compensated algorithm to compute the ESFs based on error-free transformations and classic summation algorithm. The result computed by our method is roughly as accurate as the one computed by the classic summation algorithm using twice the working precision, with a final rounding to the working precision (see Theorem 4).

3.1. Compensated algorithm

We present hereafter a compensated scheme to evaluate the *k*th elementary symmetric function.

Algorithm 2: Compensated Summation Algorithm

```
**Input:** $X = (x_1, \dots, x_n)$ and $k$
**Output:** $k$-th ESF $\overline{S}_k^{(n)}(X) = \overline{S}_k^{(n)}$
function $\overline{S}_k^{(n)}$=CompSumESF$(X, k)$
$\widehat{S}_0^{(i)} = 1,\ 1 \le i \le n-1;\ \widehat{S}_j^{(i)} = 0,\ j > i;\ \widehat{S}_1^{(1)} = x_1;$
$\widehat{\epsilon S}_j^{(i)} = 0, \forall\, i, j$
for $i = 2 : n$
    for $j = \max\{1, i+k-n\} : \min\{i, k\}$
        $[p, \beta_j^{(i)}] =$ TwoProd$(x_i, \widehat{S}_{j-1}^{(i-1)})$;
        $[\widehat{S}_j^{(i)}, \sigma_j^{(i)}] =$ TwoSum$(\widehat{S}_j^{(i-1)}, p)$;
        $\widehat{\epsilon S}_j^{(i)} = \widehat{\epsilon S}_j^{(i-1)} \oplus (\beta_j^{(i)} \oplus \sigma_j^{(i)}) \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)}$
    end
end
$\overline{S}_k^{(n)} = \widehat{S}_k^{(n)} \oplus \widehat{\epsilon S}_k^{(n)}$
```

From Algorithm 2 and Theorem 1, it follows that

$$p + \beta_j^{(i)} = x_i \times \widehat{S}_{j-1}^{(i-1)} \text{ and } \widehat{S}_j^{(i)} + \sigma_j^{(i)} = \widehat{S}_j^{(i-1)} + p, \tag{11}$$

and then

$$\widehat{S}_j^{(i)} + (\beta_j^{(i)} + \sigma_j^{(i)}) = \widehat{S}_j^{(i-1)} + x_i \times \widehat{S}_{j-1}^{(i-1)}. \tag{12}$$

Let $\epsilon S_j^{(i)}$ be the error between the theoretical result and the computed one, so that

$$\widehat{S}_j^{(i)} + \epsilon S_j^{(i)} = S_j^{(i)},\ \forall\, i, j. \tag{13}$$

Since

$$S_j^{(i)} = S_j^{(i-1)} + x_i \times S_{j-1}^{(i-1)}, \tag{14}$$

and by (12–14), we can deduce that

$$\epsilon S_j^{(i)} = \epsilon S_j^{(i-1)} + (\beta_j^{(i)} + \sigma_j^{(i)}) + x_i \times \epsilon S_{j-1}^{(i-1)}. \tag{15}$$

Therefore, computing an approximate $\widehat{\epsilon S}_k^{(n)}$ of $\epsilon S_k^{(n)}$ in the working precision and correcting the original result $\widehat{S}_k^{(n)}$ with it will be expected to improve the global accuracy. The discussion below exhibits the validation of Algorithm 2.

3.2. Forward error bound

Lemma 1. *Let us consider the recurrence (15), with the notation $w_j^{(i)} = \beta_j^{(i)} + \sigma_j^{(i)}$ and Definition (1). Then, we have*

$$\epsilon S_k^{(n)} = \sum_{i=2}^{n} \sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} w_j^{(i)} S_{k-j}^{(n-i)}(x_{i+1}, \dots, x_n). \tag{16}$$

Proof. The proof is direct by induction. We can also obtain the result by drawing data dependency graph like in [20], [21] and [22]. Considering (15), we deem that the final result $\epsilon S_k^{(n)}$ of the loop consists of the roundoff errors $w_j^{(i)} = \beta_j^{(i)} + \sigma_j^{(i)}$ generated in every step of Algorithm 2. The contribution of $w_j^{(i)}$ can be easy to obtain with the data dependency graph. Let $S(i, j) = \epsilon S_j^{(i)}$, $V\_arr(i, j) = 1$, $D\_arr(i, j) = x_i$ and $C(i, j) = w_j^{(i)}$ for $1 \le j \le i \le n$ in (10). We find that there are $\binom{n-i}{k-j}$ paths from the node (i, j) to node (n, k), and each path consists of $k - j$ diagonal arrows and $(n - i) - (k - j)$ vertical arrows, hence, we deduce that the coefficients of $w_j^{(i)}$ should be an elementary symmetric function $S_{k-j}^{(n-i)}(x_{i+1} \dots x_n)$. Take into account that $w_j^{(1)} = 0$, we obtain the expected result (16). □

Lemma 2. *For a vector of n floating-point numbers $X = (x_1, \ldots, x_n)$, Algorithm 2 computes the evaluation of $\epsilon S_k^{(n)}$ defined in Lemma 1. Then the computed result $\widehat{\epsilon S}_k^{(n)}$ satisfies the following forward error bound,*

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le 2u(n-1)(1+u)\gamma_{2(n-1)}(1+\gamma_{2(n-2)})S_k^{(n)}(|X|),$$

where $S_k^{(n)}(|X|) = S_k^{(n)}(|x_1|, \ldots, |x_n|)$.

Proof. First, we will present the expression of $\widehat{\epsilon S}_j^{(i)}$. By Algorithm 2, let $\widehat{w}_j^{(i)} = \beta_j^{(i)} \oplus \sigma_j^{(i)} = w_j^{(i)}(1+\delta),\ |\delta| < u$, we have

$$\widehat{\epsilon S}_j^{(i)} = \widehat{\epsilon S}_j^{(i-1)} \oplus \widehat{w}_j^{(i)} \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)} = \widehat{w}_j^{(i)}{<}2{>} + \widehat{\epsilon S}_j^{(i-1)}{<}2{>} + x_i \times \widehat{\epsilon S}_{j-1}^{(i-1)}{<}2{>}.$$

And in particular, the initial and boundary values may need some modifications, such as

$$\widehat{\epsilon S}_1^{(i)} = \widehat{\epsilon S}_1^{(i-1)} \oplus \widehat{w}_1^{(i)} = \widehat{\epsilon S}_1^{(i-1)}{<}1{>} + \widehat{w}_1^{(i)}{<}1{>},$$
$$\widehat{\epsilon S}_i^{(i)} = \widehat{w}_i^{(i)} \oplus x_i \otimes \widehat{\epsilon S}_{i-1}^{(i-1)} = \widehat{w}_i^{(i)}{<}1{>} + x_i \times \widehat{\epsilon S}_{i-1}^{(i-1)}{<}2{>}$$

with $\widehat{w}_1^{(i)} = \sigma_1^{(i)}$ and $\widehat{w}_i^{(i)} = \beta_i^{(i)}$.

Like in the proof of Lemma 1, we have

$$\widehat{\epsilon S}_k^{(n)} = \sum_{i=2}^{n} \sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} w_j^{(i)} \times S_{k-j}^{(n-i)}(x_{i+1}, \ldots, x_n)(1+\theta(i,j)),$$

where $|\theta(i,j)| < \gamma_{2(n-1)}$, which can be easily proved by induction and directly obtained with the data dependency graph. Then we obtain

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \gamma_{2(n-1)} \sum_{i=2}^{n} \sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} |w_j^{(i)}| \times S_{k-j}^{(n-i)}(|x_{i+1}|, \ldots, |x_n|). \tag{17}$$

Next, we will obtain the bound of $|w_j^{(i)}|$. From Theorem 1, we get

$$|\beta_j^{(i)}| \le u|x_i \times \widehat{S}_{j-1}^{(i-1)}| \text{ and } |\sigma_j^{(i)}| \le u|\widehat{S}_j^{(i-1)} + x_i \otimes \widehat{S}_{j-1}^{(i-1)}|,$$

and then

$$|w_j^{(i)}| \le |\beta_j^{(i)}| + |\sigma_j^{(i)}| \le 2u(1+u)(|\widehat{S}_j^{(i-1)}| + |x_i| \times |\widehat{S}_{j-1}^{(i-1)}|). \tag{18}$$

Taking into account (9), we have

$$|\widehat{S}_j^{(i-1)}| \le (1+\gamma_{2(i-2)})S_j^{(i-1)}(|x_1|, \ldots, |x_{i-1}|),$$
$$|\widehat{S}_{j-1}^{(i-1)}| \le (1+\gamma_{2(i-2)}) \times S_{j-1}^{(i-1)}(|x_1|, \ldots, |x_{i-1}|). \tag{19}$$

Then, considering (18), (19) and

$$S_j^{(i-1)}(|x_1|, \ldots, |x_{i-1}|) + |x_i| \times S_{j-1}^{(i-1)}(|x_1|, \ldots, |x_{i-1}|) = S_j^{(i)}(|x_1|, \ldots, |x_i|),$$

we obtain

$$|w_j^{(i)}| \le 2u(1+u)(1+\gamma_{2(n-2)})S_j^{(i)}(|x_1|, \ldots, |x_i|). \tag{20}$$

Since

$$\sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} S_j^{(i)}(|x_1|, \ldots, |x_i|)S_{k-j}^{(n-i)}(|x_{i+1}|, \ldots, |x_n|) = S_k^{(n)}(|x_1|, \ldots, |x_n|),$$

then from (17), (20), we can finally deduce the expected result. □

Theorem 4. *For a vector of n floating-point numbers $X = (x_1, \ldots, x_n)$, the relative forward error bound in Algorithm 2 satisfies*

$$\left|\frac{\overline{S}_k^{(n)} - S_k^{(n)}}{S_k^{(n)}}\right| \le u + \frac{1}{k}\gamma_{2(n-1)}^2 \mathtt{cond}(S_k^{(n)}(X)),$$
$$\left|\frac{\widehat{S}_1^{(n)} - S_1^{(n)}}{S_1^{(n)}}\right| \le u + \gamma_{n-1}^2 \mathtt{cond}(S_1^{(n)}),$$

$$\left|\frac{\widehat{S}_n^{(n)} - S_n^{(n)}}{S_n^{(n)}}\right| \le u + \frac{1}{n}\gamma_n\gamma_{2n}\texttt{cond}(S_n^{(n)}),$$

with $2 \le k \le n-1$, $k = 1$, $k = n$, *respectively.*

Proof. For $2 \le k \le n-1$, by Algorithm 2, we have

$$\overline{S}_k^{(n)} = \widehat{S}_k^{(n)} \oplus \widehat{\epsilon S}_k^{(n)} = (\widehat{S}_k^{(n)} + \widehat{\epsilon S}_k^{(n)})(1+\delta) = (\widehat{S}_k^{(n)} + \epsilon S_k^{(n)} - \epsilon S_k^{(n)} + \widehat{\epsilon S}_k^{(n)})(1+\delta),$$

with $|\delta| < u$. Considering $\widehat{S}_k^{(n)} + \epsilon S_k^{(n)} = S_k^{(n)}$, we have

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le u|S_k^{(n)}| + (1+u)|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}|.$$

Taking into account that $(1+u)^2(1+\gamma_{2(n-2)}) \le (1+\gamma_{2(n-1)})$ and $2(n-1)u(1+\gamma_{2(n-1)}) = \gamma_{2(n-1)}$, from Lemma 2, we obtain

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le u|S_k^{(n)}| + \gamma_{2(n-1)}^2 S_k^{(n)}(|X|). \tag{21}$$

At last the desired bound follows from the definition of condition number (8). For the cases of $k = 1$ and $k = n$, the results come from Proposition 4.5 of [7] and Theorem 2 of [3], respectively. □

3.3. Running error analysis

In practical calculations, it is desirable to obtain a corresponding error bound together with the computed value. The *a priori* error bound (21) of Theorem 4 is entirely adequate for theoretical purposes, but lakes sharpness. For this requirement, we perform a running error analysis of `CompSumESF`, which provides a sharper and *a posteriori* error bound. We first provide the compensated summation algorithm with running error bound, denoted by `CompSumESFwErr`. Then we prove the efficiency and rationality of this algorithm with Theorem 5.

Lemma 3. *For a vector of n floating-point numbers* $X = (x_1, \dots, x_n)$, $\epsilon S_k^{(n)}$ *is the theoretical result and* $\widehat{\epsilon S}_k^{(n)}$ *is the corresponding numerical result in floating-point arithmetic computed by Algorithm 2, then*

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le fl\left(\frac{\widehat{\gamma}_{2(n-1)} \otimes \widehat{ES}_k^{(n)}}{1-3nu}\right) := \hat{\alpha} \tag{22}$$

where $\widehat{ES}_k^{(n)}$ *is computed with the recurrence relation*

$$\widehat{ES}_j^{(i)} = \widehat{ES}_j^{(i-1)} \oplus |\beta_j^{(i)} \oplus \sigma_j^{(i)}| \oplus |x_i| \otimes \widehat{ES}_{j-1}^{(i-1)} \tag{23}$$

by Algorithm 3 in floating-point arithmetic and $3nu < 1$.

Algorithm 3: Compensated Summation Algorithm with running error bound

```
Input: $X = (x_1, \dots, x_n)$ and $k$
Output: k-th ESF $\overline{S}_k^{(n)}(X) = \overline{S}_k^{(n)}$ and Running Error Bound $\mu$
function $[\overline{S}_k^{(n)}, \mu]$=CompSumESFwErr$(X, k)$
$\widehat{S}_0^{(i)} = 1,\ 1 \le i \le n-1;\ \widehat{S}_j^{(i)} = 0,\ j > i;\ \widehat{S}_1^{(1)} = x_1;$
$\widehat{\epsilon S}_j^{(i)} = 0, \widehat{ES}_j^{(i)} = 0, \forall\, i, j$
for $i = 2 : n$
    for $j = \max\{1, i+k-n\} : \min\{i, k\}$
      $[p, \beta_j^{(i)}]$ = TwoProd$(x_i, \widehat{S}_{j-1}^{(i-1)})$;
      $[\widehat{S}_j^{(i)}, \sigma_j^{(i)}]$ = TwoSum$(\widehat{S}_j^{(i-1)}, p)$;
      $\widehat{\epsilon S}_j^{(i)} = \widehat{\epsilon S}_j^{(i-1)} \oplus (\beta_j^{(i)} \oplus \sigma_j^{(i)}) \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)}$
      $\widehat{ES}_j^{(i)} = \widehat{ES}_j^{(i-1)} \oplus |\beta_j^{(i)} \oplus \sigma_j^{(i)}| \oplus |x_i| \otimes \widehat{ES}_{j-1}^{(i-1)}$
    end
end
$[\overline{S}_k^{(n)}, c]$ = FastTwoSum$(\widehat{S}_k^{(n)}, \widehat{\epsilon S}_k^{(n)})$
$\hat{\alpha} = (\widehat{\gamma}_{2(n-1)} \otimes \widehat{ES}_k^{(n)}) \oslash (1-3nu)$
$\mu = (|c| \oplus \hat{\alpha}) \oslash (1-2u)$
```

Proof. From (17) in Lemma 2, we have

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \gamma_{2(n-1)} ES_k^{(n)}, \tag{24}$$

with

$$ES_k^{(n)} = \sum_{i=2}^{n} \sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} |w_j^{(i)}| S_{k-j}^{(n-i)}(|x_{i+1}|, \ldots, |x_n|),$$

where $w_j^{(i)} = \beta_j^{(i)} + \sigma_j^{(i)}$. Obviously $ES_k^{(n)}$ can be derived from the following recurrence

$$ES_j^{(i)} = ES_j^{(i-1)} + |\beta_j^{(i)} + \sigma_j^{(i)}| + |x_i| \times ES_{j-1}^{(i-1)}$$

with the initial values $ES_j^{(i)} = 0, \forall\, i, j$.

From (23) and the model (3), we have

$$\widehat{ES}_j^{(i)} = \{[\widehat{ES}_j^{(i-1)} + |\beta_j^{(i)} + \sigma_j^{(i)}| \frac{1}{1+\delta_1}] \frac{1}{1+\delta_2} + |x_i| \times \widehat{ES}_{j-1}^{(i-1)} \frac{1}{1+\delta_3}\} \frac{1}{1+\delta_4},$$

with $|\delta_t| \le u$, for $t = 1, \ldots, 4$. Then taking into account $\widehat{ES}_j^{(i)} \ge 0$, we obtain

$$(1+u)^3 \widehat{ES}_j^{(i)} \ge \widehat{ES}_j^{(i-1)} + |\beta_j^{(i)} + \sigma_j^{(i)}| + |x_i| \times \widehat{ES}_{j-1}^{(i-1)}.$$

By induction we can prove that

$$(1+u)^{3(n-1)} \widehat{ES}_k^{(n)} \ge ES_k^{(n)}. \tag{25}$$

Finally, by the first part in (4) and (25), it follows from (24) that

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le (1+u)^{3(n-1)+2} \widehat{ES}_k^{(n)} \otimes \widehat{\gamma}_{2(n-1)}.$$

Using the second relation in (4), we obtain the expected error bound (22). □

Theorem 5. *Assume* $3nu < 1$, *then a running error bound of Algorithm 2 is given by*

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le fl(\frac{|c| \oplus \widehat{\alpha}}{1-2u}) := \mu, \tag{26}$$

where $\widehat{\alpha}$ *is the error bound defined by (22) in Lemma 3 and c is obtained by* $[\overline{S}_k^{(n)}, c] = \texttt{FastTwoSum}(\widehat{S}_k^{(n)}, \widehat{\epsilon S}_k^{(n)})$.

Proof. From Theorem 1, we have $\overline{S}_k^{(n)} + c = \widehat{S}_k^{(n)} + \widehat{\epsilon S}_k^{(n)}$. Considering $S_k^{(n)} = \widehat{S}_k^{(n)} + \epsilon S_k^{(n)}$, we can deduce that

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le |\overline{S}_k^{(n)} - (\widehat{S}_k^{(n)} + \widehat{\epsilon S}_k^{(n)})| + |\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| = |c| + \widehat{\alpha} \le (1+u)(|c| \oplus \widehat{\alpha}) \le fl(\frac{|c| \oplus \widehat{\alpha}}{1-2u}).$$

□

In order to obtain a running relative error bound, when $\overline{S}_k^{(n)} \ne 0$, we may deduce by (26) that

$$\frac{|\overline{S}_k^{(n)} - S_k^{(n)}|}{|S_k^{(n)}|} \le \frac{\mu}{|S_k^{(n)}|} \le \frac{\mu}{|\overline{S}_k^{(n)}(1 - \frac{\mu}{\overline{S}_k^{(n)}})|} \le \frac{\mu}{|\overline{S}_k^{(n)}|} + \mathcal{O}((\frac{\mu}{|\overline{S}_k^{(n)}|})^2). \tag{27}$$

Other techniques, such as interval arithmetic and basically using rounding upward for all the operations, have been used to compute a rigorous error bound. Compared with these techniques, our method may provide a more sharper bound together with the computed result, without requiring the additional significant computational cost.

3.4. Double–double library

It is interesting to compare our compensated algorithm with other approaches to obtain high-precision. A standard way is by using multiple precision libraries such as ARPREC [24], MP [25], and MPFR [26]. But if we just want to double the IEEE-754 double precision, a more efficient way is to use Bailey's double–double arithmetic [27]. A double–double number a is the pair (ah, al) of floating-point numbers with

$$a = ah + al, \ \ |al| < u|ah| \text{ and } |al| < u|a|. \tag{28}$$

In the sequel, we present two algorithms to compute the addition of two double–double numbers and a double number times a double–double number shown in Table 2.

Using internally double–double numbers, we can now propose another accurate algorithm as follows.

Table 2
Addition and multiplication in double–double format.

Algorithm	Flops	Ref.
$[rh, rl] = \texttt{add\_dd\_dd}(ah, al, bh, bl)$	20	[28]
$[rh, rl] = \texttt{prod\_dd\_d}(ah, al, b)$	22	[28,29]
$[rh, rl] = \texttt{add\_cdd\_cdd}(ah, al, bh, bl)$	40	[27]
$[rh, rl] = \texttt{prod\_cdd\_cd}(ah, al, b)$	128	[27]

Algorithm 4: Accurate Summation Algorithm in Double-Double Format

```
Input: $X = (x_1, \ldots, x_n)$ and $k$
Output: k-th ESF $S_k^{(n)}(X) = S_k^{(n)} = Sh_k^{(n)}$
function $[Sh_k^{(n)}, Sl_k^{(n)}]$=DDSumESF$(X, k)$
$Sh_0^{(i)} = 1,\ 1 \le i \le n-1;\ Sh_j^{(i)} = 0,\ j > i;\ Sh_1^{(1)} = x_1;$
$Sl_j^{(i)} = 0, \forall\ i, j$
For $i = 2 : n$
    For $j = \max(1, i+k-n) : \min\{i, k\}$
      $[rh, rl] = \texttt{prod\_dd\_d}(Sh_{j-1}^{(i-1)}, Sl_{j-1}^{(i-1)}, x_i);$
      $[Sh_j^{(i)}, Sl_j^{(i)}] = \texttt{add\_dd\_dd}(rh, rl, Sh_j^{(i-1)}, Sl_j^{(i-1)});$
    end
end
```

Before studying the accuracy of Algorithm 4 , let us recall the model of double–double arithmetic, the similar results and proof can be seen in [30].

Remark 1 ([28, p.182]). For a standard model of floating-point arithmetic for the double–double algorithms

$$fl(a \odot b) = (a \odot b)(1 + \delta), \tag{29}$$

where a, b are in double–double format, $\odot \in \{+, -, \times, /\}$, and δ is bounded as follows

$$|\delta| \le u_{dd}, \text{ for } \odot \in \{+, -\};\ \ |\delta| \le 2u_{dd}, \text{ for } \odot \in \{\times, /\} \tag{30}$$

where $u_{dd} = 2u^2 = 2^{-105}$ is the roundoff unit in double–double arithmetic.

Theorem 6. *The values $\widehat{Sh}_k^{(n)}$ and $\widehat{Sl}_k^{(n)}$ returned by Algorithm 4 in floating-point arithmetic satisfy*

$$\frac{|\widehat{Sh}_k^{(n)} - S_k^{(n)}|}{|S_k^{(n)}|} \le u + \frac{1}{k}(1+u)\overline{\gamma}_{3(n-1)}\texttt{cond}(S_k^{(n)}(X)),$$

where

$$\overline{\gamma}_m = \frac{mu_{dd}}{(1 - mu_{dd})}. \tag{31}$$

Proof. By induction and Remark 1, we can obtain

$$\widehat{Sh}_k^{(n)} + \widehat{Sl}_k^{(n)} = S_k^{(n)} + \sum_{1 \le \pi_1 < \ldots < \pi_k \le n} x_{\pi_1} x_{\pi_2} \ldots x_{\pi_k} \overline{\theta}_t^{(\pi_1 \ldots \pi_k)},$$

where $|\overline{\theta}_t^{(\pi_1 \ldots \pi_k)}| \le \overline{\gamma}_{3(n-1)}$. Then considering the property of a double–double number $|\widehat{Sl}_k^{(n)}| < u|\widehat{Sh}_k^{(n)} + \widehat{Sl}_k^{(n)}|$ we have

$$\begin{aligned}|\widehat{Sh}_k^{(n)} - S_k^{(n)}| &\le |\widehat{Sl}_k^{(n)}| + |\widehat{Sh}_k^{(n)} + \widehat{Sl}_k^{(n)} - S_k^{(n)}| \le u|\widehat{Sh}_k^{(n)} + \widehat{Sl}_k^{(n)}| + \overline{\gamma}_{3(n-1)} S_k^{(n)}(|X|) \\ &\le u|S_k^{(n)}| + (1+u)\overline{\gamma}_{3(n-1)} S_k^{(n)}(|X|).\end{aligned}$$

Finally, by the definition of condition number we deduce the desired error bound. □

4. In complex arithmetic case

We have already proposed a compensated summation algorithm to evaluate the kth ESF with real floating-point inputs and given the corresponding error analysis. In this section we will consider the case of complex floating-point inputs.

4.1. CompSumESF in complex version

For the input $X = (x_1, \dots, x_n),\ x_i \in \mathbb{F} + i\mathbb{F}$, using complex error-free transformations `TwoSumCplx` and `TwoProdCplx` instead of `TwoSum` and `TwoProd` in the inner loop of Algorithm 2 (`CompSumESF`), we will get the compensated summation algorithm in complex arithmetic case denoted by `CompSumESFCplx`. Here for error analysis, there is no difference to use either `TwoProdCplxSingleSpliting` or `TwoProdCplx`.

$$\begin{aligned}
[p, e_j^{(i)}, f_j^{(i)}, g_j^{(i)}] &= \texttt{TwoProdCplx}(x_i, \widehat{S}_{j-1}^{(i-1)}); \\
[\widehat{S}_j^{(i)}, \sigma_j^{(i)}] &= \texttt{TwoSumCplx}(\widehat{S}_j^{(i-1)}, p); \\
\widehat{w}_j^{(i)} &= \mathrm{AccSum}(e_j^{(i)}, f_j^{(i)}, g_j^{(i)}, \sigma_j^{(i)}); \\
\widehat{\epsilon S}_j^{(i)} &= \widehat{\epsilon S}_j^{(i-1)} \oplus \widehat{w}_j^{(i)} \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)};
\end{aligned} \tag{32}$$

Thanks to `AccSum` algorithm in [7], we can compute

$$w_j^{(i)} = e_j^{(i)} + f_j^{(i)} + g_j^{(i)} + \sigma_j^{(i)} \tag{33}$$

accurately. Here we stress that `AccSum` can deal with complex floating-point numbers. According to the property of the `AccSum` algorithm, we have

$$\widehat{w}_j^{(i)} = w_j^{(i)}(1+\delta),\ |\delta| < u < u_c. \tag{34}$$

Theorem 7. *For a vector of n complex floating-point numbers $X = (x_1, \dots, x_n)$, the relative forward error bound of* `CompSumESFCplx` *satisfies*

$$\left|\frac{\overline{S}_k^{(n)} - S_k^{(n)}}{S_k^{(n)}}\right| \le u + \frac{1}{k}\widetilde{\gamma}_{2(n-1)}^2 \mathtt{cond}(S_k^{(n)}(X)),$$

$$\left|\frac{\widehat{S}_1^{(n)} - S_1^{(n)}}{S_1^{(n)}}\right| \le u + \gamma_{n-1}^2 \mathtt{cond}(S_1^{(n)}),$$

$$\left|\frac{\widehat{S}_n^{(n)} - S_n^{(n)}}{S_n^{(n)}}\right| \le u + \frac{1}{n}\widetilde{\gamma}_n\widetilde{\gamma}_{2n} \mathtt{cond}(S_n^{(n)}),$$

with $2 \le k \le n-1,\ k = 1,\ k = n$, respectively.

Proof. For $2 \le k \le n-1$, let

$$\beta_j^{(i)} = e_j^{(i)} + f_j^{(i)} + g_j^{(i)}. \tag{35}$$

From (33) we will have $w_j^{(i)} = \beta_j^{(i)} + \sigma_j^{(i)}$. Then from complex error-free transformations Theorem 2 and (32), we will obtain $|\beta_j^{(i)}| \le u_c |x_i \times \widehat{S}_{j-1}^{(i-1)}|$.

Now, with the properties (6) in complex absolute value, we can perform a straightforward adaptation of the proof of Lemma 2 by substituting u_c and $\widetilde{\gamma}_k$ for u and γ_k respectively. Then we obtain the similar result to that of Lemma 2.

Finally, we use the fore part of the proof in Theorem 4, change the second part of the error bound with $u < u_c$. Then with the condition number in the complex arithmetic case in Section 2.3, we obtain the desired bound.

For the case of $k = 1$, it is not necessary to use u_c, then we deem that the proof can be as the same as that in real arithmetic case. The proposition 4.2 in [5] used the result in [7] to obtain the error bound. But it will get a sharper error bound to use the similar proof just like that in [7]. For the case of $k = n$, the proof is very similar by referring to [3]. □

4.2. CompSumESFwErr in complex version

Next, we will consider the running error bound of `CompSumESFCplx` and present its implement, denoted by `CompSumESFwErrCplx`. We need add

$$\widehat{ES}_j^{(i)} = \widehat{ES}_j^{(i-1)} \oplus |\widehat{w}_j^{(i)}| \oplus |x_i| \otimes \widehat{ES}_{j-1}^{(i-1)} \tag{36}$$

after (32) and let all of them in the inner loop. We also need modify the tail of `CompSumESFwErr` with

$$\begin{aligned}
[\overline{S}_k^{(n)}, c] &= \texttt{TwoSumCplx}(\widehat{S}_k^{(n)}, \widehat{\epsilon S}_k^{(n)}) \\
\widehat{\alpha}_c &= (\widehat{\widetilde{\gamma}}_{2(n-1)} \otimes \widehat{ES}_k^{(n)}) \oslash (1 - (9n-6)u) \\
\mu &= (|c| \oplus \widehat{\alpha}_c) \oslash (1 - 2u)
\end{aligned} \tag{37}$$

Theorem 8. *For a vector of n complex floating-point numbers* $X=(x_1,\dots,x_n)$, *assume* $nu \ll 1$, *the running error bound of* `CompSumESFwErrCplx` *is given by*

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le \mu.$$

where the main parts of `CompSumESFwErrCplx` *consist of (32), (36) and (37).*

Proof. Substituting u_c and $\widetilde{\gamma}_k$ for u and γ_k, respectively, we will get

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \widetilde{\gamma}_{2(n-1)} ES_k^{(n)}. \tag{38}$$

where $ES_k^{(n)}$ is as the same as that in Lemma 3 but performed in complex arithmetic. It should be stressed that $\widehat{ES}_j^{(i)}$ in (36) is real floating-point number. Then considering (35), with the similar proof of Lemma 3 we also can obtain (25).

From $u_c=\sqrt{2}\gamma_2$, we have $\widehat{u_c}=u_c(1+\theta_3)$ and $\gamma_2<u_c<\gamma_3$. Then it is easy to get $\widetilde{\gamma}_k<\gamma_{3(k+1)}$. We can deduce that $\widehat{\widetilde{\gamma}}_k = fl(\frac{k\widehat{u_c}}{1-k\widehat{u_c}}) = \frac{(1+\theta_6)ku_c}{1-(1+\theta_4)ku_c} > \frac{1-\gamma_6}{1+\gamma_4\widetilde{\gamma}_k}\widetilde{\gamma}_k$. Considering that $\frac{1-\gamma_6}{1+\gamma_4\widetilde{\gamma}_k} > \frac{1-\gamma_6}{1+\gamma_4\gamma_{3(k+1)}} > \frac{1}{1+\gamma_{3k+1}}$ for $k>1$, then we have $\widetilde{\gamma}_k < (1+u)^{3k+2}\widehat{\widetilde{\gamma}}_k$. Hence $\widetilde{\gamma}_{2(n-1)} < (1+u)^{6n-4}\widehat{\widetilde{\gamma}}_{2(n-1)}$. Then from (38) and (25) we have

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \widehat{\alpha_c} \tag{39}$$

where $\widehat{\alpha_c}$ is defined in (37).

Finally, from Theorem 2 we can directly adopt the proof in Theorem 5 and obtain the expected result. □

4.3. DDSumESF in complex version

At the end of this section, we will consider the complex double–double format, which is discussed in [27]. If the pair (ah, al) satisfies $ah, al \in \mathbb{F}+i\mathbb{F}$ and the properties (28), then $a=ah+al$ is a complex double–double number. For a, b in complex double–double format, the basic arithmetic operation $\odot\in\{+,-,\times\}$ computed under the standard model (29) satisfy

$$|\delta| \le u_{dd},\ for \odot \in\{+,-\};\ \ |\delta| \le \sqrt{2}\gamma_4,\ for \odot\in\{\times\}, \tag{40}$$

where the bound $\sqrt{2}\gamma_4$ is similar to (31) but obtained by using $2u_{dd}$ as the unit in (5).

Using the following iterations substituting in the inner loop of the Algorithm 4, we get `DDSumESFCplx`.

$$\begin{aligned} [rh, rl] &= \texttt{prod\_cdd\_cd}(Sh_{j-1}^{(i-1)}, Sl_{j-1}^{(i-1)}, x_i);\\ [Sh_j^{(i)}, Sl_j^{(i)}] &= \texttt{add\_cdd\_cdd}(rh, rl, Sh_j^{(i-1)}, Sl_j^{(i-1)}); \end{aligned}$$

where `prod_cdd_cd` and `add_cdd_cdd` are shown in Table 2. Then the error bound of `DDSumESFCplx` satisfies

$$\frac{|\widehat{Sh}_k^{(n)} - S_k^{(n)}|}{|S_k^{(n)}|} \le u + \frac{1}{k}(1+u)\widetilde{\widetilde{\gamma}}_{2(n-1)}\,\texttt{cond}(S_k^{(n)}(X)), \tag{41}$$

where $\widetilde{\widetilde{\gamma}}_{2(n-1)} = \frac{2(n-1)u_{cdd}}{1-2(n-1)u_{cdd}}$, with $u_{cdd}=\sqrt{2}\gamma_4$.

Based on the discussion above we can directly deduce this error bound referring to the proof of Theorem 6.

5. Computing characteristic polynomials from perturbed eigenvalues

In this section, we will discuss the role of high precision summation algorithm, such as `CompSumESF`, in computing characteristic polynomials from perturbed eigenvalues. The following theorem indicates that, if the relative perturbation bounds are smaller than the working precision, the roundoff error bound for the coefficients of the polynomial are similar to the perturbation bounds.

Theorem 9. *Let* $\tilde{x}_i = x_i(1+\epsilon_i)$ *be eigenvalues of a matrix whose characteristic polynomial is* $\sum_{i=0}^{n}\tilde{c}_i x^{n-i}$, $c_0=\tilde{c}_0=1$ *and let* $\epsilon_{rel} = \max_{1\le i\le n}|\epsilon_i|$. *Denote by* $fl[\tilde{c}_k]$ *the coefficients computed by* `CompSumESF` *in floating-point arithmetic from the perturbed eigenvalues* $\tilde{x}_i$. *if* x_i, $\tilde{x}_i$ *are real,* $n\epsilon_{rel}<1$, $2\le k\le n$ *and* $c_k\ne 0$, *then*

$$\frac{|fl[\tilde{c}_k] - c_k|}{|c_k|} \le u + \frac{1}{k}\frac{\gamma_{2(n-1)}^2 + k\epsilon_{rel}(1+u)}{1-k\epsilon_{rel}}\texttt{cond}(S_k^{(n)}(X)) \tag{42}$$

Proof. First we have the following triangle inequality

$$\frac{|fl[\tilde{c}_k] - c_k|}{|c_k|} \le \frac{|fl[\tilde{c}_k] - \tilde{c}_k|}{|c_k|} + \frac{|\tilde{c}_k - c_k|}{|c_k|}. \tag{43}$$

Applying the perturbation bound in Theorem 2.13 in [2] to the second summand gives

$$\frac{|\tilde{c}_k - c_k|}{|c_k|} \leq \frac{k\epsilon_{rel}}{1-k\epsilon_{rel}} \frac{S_k^{(n)}(|X|)}{|c_k|} = \frac{\epsilon_{rel}}{1-k\epsilon_{rel}} \texttt{cond}(S_k^{(n)}(X)). \tag{44}$$

To the first summand, from the error bound (21) in Theorem 4, we can deduce

$$|fl[\tilde{c}_k] - \tilde{c}_k| \leq u|S_k^{(n)}(\tilde{X})| + \gamma_{2(n-1)}^2 S_k^{(n)}(|\tilde{X}|), \tag{45}$$

where $\tilde{X} = (\tilde{x}_1, \ldots, \tilde{x}_n)$. Since $\frac{||\tilde{x}_i|-|x_i||}{|x_i|} \leq \frac{|\tilde{x}_i - x_i|}{|x_i|} \leq \epsilon_{rel}$, from Corollary 2.10 and Corollary 2.11 in [2], we have

$$\begin{aligned} |S_k^{(n)}(\tilde{X})| &\leq |S_k^{(n)}(X)| + \frac{k\epsilon_{rel}}{1-k\epsilon_{rel}} S_k^{(n)}(|X|) \\ S_k^{(n)}(|\tilde{X}|) &\leq \frac{S_k^{(n)}(|X|)}{1-k\epsilon_{rel}}. \end{aligned} \tag{46}$$

From (2) we have the relationship between the coefficient and the elementary symmetric function $c_k = (-1)^k S_k^{(n)}(X)$. Then by By (43–46) and the definition of condition number (8), we can obtain the expected result (42). □

Theorem 9 implies that it can not be expected to compute accurate coefficients from the ill-conditioned eigenvalues, even though applying the high precision summation algorithm (such as CompSumESF). For instance, when $1 > k\epsilon_{rel} \geq \mathcal{O}(\gamma_{2n}) \approx \mathcal{O}(2nu)$, the error bound (42) will be simplified to

$$\frac{|fl[\tilde{c}_k] - c_k|}{|c_k|} \leq \mathcal{O}(\epsilon_{rel}) \texttt{cond}(S_k^{(n)}(X)).$$

Therefore, the accuracy of the computed coefficients depends strongly on the accuracy of the initial perturbed eigenvalues. Only when the eigenvalues are not perturbed or accurate enough, such as $k\epsilon_{rel} \approx \mathcal{O}(\gamma_{2n}^2)$, the high precision summation algorithm will show its effectiveness.

The whole conclusion above is based on the assumption that Eq. (44) is not much more pessimistic. However, for instance when the eigenvalues are clustered or uniformly distributed (in Section 6.3), even though the condition number is large, the relative perturbation $|\tilde{c}_k - c_k|/|c_k|$ is still small in the level of $\mathcal{O}(\epsilon_{rel})$. Hence, in this case high precision summation algorithm will be necessary to compute accurate coefficients.

In the case of complex eigenvalues, we will get the similar results.

6. Numerical test

In this section we present timing and accuracy results. All numerical experiments are performed in IEEE-754 double precision as working precision. We compare the algorithms in complex floating-point arithmetic (SumESFCplx, CompSumESFCplx, CompSumESFwErrCplx and DDSumESFCplx) as well as the algorithms SumESF, CompSumESF, CompSumESFwErr and DDSumESF in real floating-point arithmetic. We also show the compensated summation algorithms in some cases are helpful in computing characteristic polynomial from matrix.

6.1. Accuracy test

All accuracy measurements are done in MATLAB 7.0. To test the accuracy of our algorithms we need extremely ill-conditioned elementary symmetric functions, hence we designed GenESF based on GenPoly [29] and GenDot [7]. The basic idea of this generation algorithm is to construct random vector X with $S_k^{(n)}(|X|) \approx \text{cond\_exp} \times |v|$ and $S_k^{(n)}(X) \approx v$. We generated the floating-point vector[4] $X \in \mathbb{F}^n$ with the dimension n from 10 to 30, k being a random integer in the interval $[2, n-1]$, and the condition number of computing kth elementary symmetric function varying from 10^4 to 10^{33}. In this part of numerical tests, the exact result is obtained by the original summation algorithm using quad-double format [31]. We also assume that the relative errors greater than 2 be the value 2 just like [7], which means there is no useful information left.

As we can see on Fig. 2, CompSumESF (Algorithm 2) exhibits the expected behavior. When the condition number is smaller than $1/u$, the relative error of CompSumESF is equal to or smaller than u. This relative error degrades to no precision at all for the condition number between $1/u$ and $1/u^2$. Meanwhile, it is shown that CompSumESF and DDSumESF nearly have the same accuracy. We also present the forward error bound of DDSumESF shown as the dashed line TheoBoundDD. In fact, DDSumESF may be a little more accurate than CompSumESF, however it is not significant from Fig. 2. It is also shown that the forward error bound from Theorem 4 is valid, but pessimistic compared with the relative running error bound from CompSumESFwErr, which is exhibited as the green real line RunErrBound. Besides of DDSumESF, we also compare our algorithm with LejaSumESF, which uses Leja ordering of the zeros in conjunction with the original summation algorithm (see [11]). However, it does not give significantly higher accuracy of the results in our numerical tests, partly due to the fact that the inputs are ordered randomly in the generation algorithm.

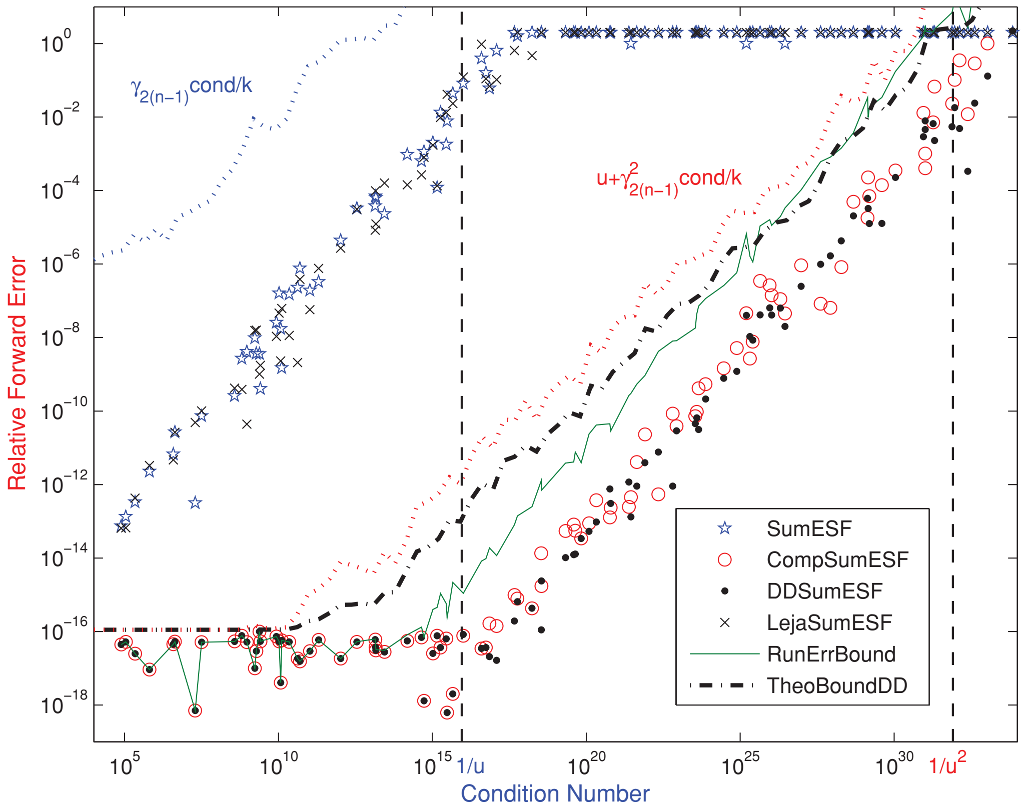

Fig. 2. Accuracy of evaluation with respect to the condition number in real floating-point arithmetic.

Similar results for the compared algorithms in complex arithmetic case are shown in Fig. 3. The conclusions are similar to the real arithmetic case.

6.2. Running time test

All timing tests are done on the following environments.

- Env 1 : Laptop with Intel Pentium Dual CPU E2160, each at 1.8GHz, and with Microsoft Visual C++ 2008 with the default compiler option /od on Windows 7.
- Env 2 : Personal computer with Intel Core i3-2130 CPU, 2 cores each at 3.4GHz, and with Microsoft Visual C++ 2012 with the default compiler option /od on Windows 7.
- Env 3 : Node of workstation with Intel Xeon E5-2670 CPU, 8 cores each at 2.60GHz, and with gcc 4.4.6 with the default compiler option -O0 on x86_64-redhat-linux 4.4.6–4.

When performing the running time tests, we optimize all algorithms in C code by reversing the computing sequence of j to reduce the required storage location. Similar technique can be seen in [11] and be used in MATLAB's `poly`. There are some other optimizations just like that in [29], such as taking some `Split` out of the recurrence. Here we do not detail them. We deem that all the computing time of these algorithms does not depend on the inputs but on the dimension n and the order k. Hence, we generate the tested random inputs in the interval $[-1, 1]$ (in the case of complex inputs, both the real and imaginary parts are random floating-point numbers in $[-1, 1]$).

We perform the tests in two cases. In Case 1, we test timing of the algorithms only computing kth ESF. Then, in Case 2 we test timing of the modification algorithms computing all ESFs simultaneously, which only change the line of code $j = \max\{1, i+k-n\} : \min\{i, k\}$ in each algorithm to $j = 1 : i$. In Case 1, the number n of the inputs varies from 10 to 30 with k from 2 to $n-1$; in Case 2, n varies from 10 to 100.

We exhibit the measured running time ratios in two cases in Tables 3 and 4, respectively. Case 2 corresponds to the application of computing the coefficients of polynomial from zeros. For simplification, we still denote these algorithms by the same names as before. In both cases, it seems that `CompSumESF` is significantly faster than `DDSumESF` while the results share the same accuracy, and that the over-cost due to the running error bound supported by `CompSumESFwErr` is quite reasonable.

[4] When n and k are larger, underflow may occur during the generation of some most ill-conditioned ESF.

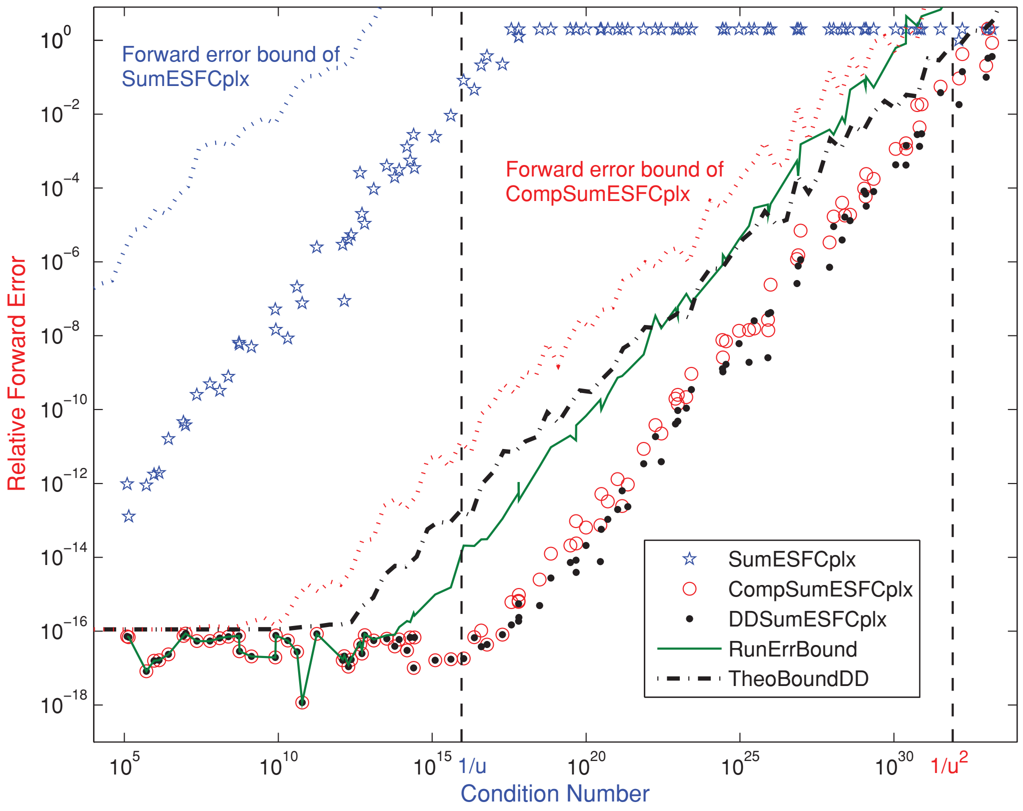

Fig. 3. Accuracy of evaluation with respect to the condition number in complex floating-point arithmetic.

Table 3
Time ratios of computing *k*th ESF with real inputs.

Case 1	`CompSumESF` / `SumESF`	`DDSumESF` / `SumESF`	`CompSumESF` / `DDSumESF`	`CompSumESF` / `CompSumESFwErr`
Env 1	3.05	5.42	57.42%	69.91%
Env 2	2.98	5.13	59.06%	72.03%
Env 3	3.51	6.66	52.80%	74.96%

Table 4
Time ratios of computing all ESFs with real inputs.

Case 2	`CompSumESF` / `SumESF`	`DDSumESF` / `SumESF`	`CompSumESF` / `DDSumESF`	`CompSumESF` / `CompSumESFwErr`
Env 1	3.91	7.48	52.97%	68.02%
Env 2	4.75	9.67	49.66%	68.23%
Env 3	3.79	7.31	51.95%	70.23%

We also consider the flop counts ratios of the algorithms in Case 2 (there are too many comparison operations in Case 1 to be suitable for flop counting). The theoretical ratio between `CompSumESF` and `SumESF` in the optimized C code is approximately 11.5, which is much larger than the corresponding running time ratios shown in Table 4. Thanks to the analysis in terms of instruction level parallelism (ILP) (see details in [29,32]), this phenomenon is surprising, but reasonable. Moreover, since the renormalization steps in `DDSumESF` may break ILP, the measured running time ratio between `CompSumESF` and `DDSumESF` is usually smaller than the theoretical one ($\approx 61\%$).

As a consequence, it seems that `CompSumESF` is a fast and accurate algorithm to compute elementary symmetric functions and can be well used in computing the coefficients of polynomial from zeros.

Similar results for the compared algorithms for complex floating-point inputs are reported in Tables 5 and 6. The conclusions are similar to the real arithmetic case.

Table 5
Time ratios of computing *k*th ESF with complex inputs.

Case 1	$\frac{\texttt{CompSumESFCplx}}{\texttt{SumESFCplx}}$	$\frac{\texttt{DDSumESFcplx}}{\texttt{SumESFCplx}}$	$\frac{\texttt{CompSumESFCplx}}{\texttt{DDSumESFCplx}}$	$\frac{\texttt{CompSumESFCplx}}{\texttt{CompSumESFwErrCplx}}$
Env 1	3.77	7.91	48.82%	72.69%
Env 2	3.68	7.62	49.39%	70.35%
Env 3	4.61	10.20	45.36%	77.76%

Table 6
Time ratios of computing all ESFs with complex inputs.

Case 2	$\frac{\texttt{CompSumESFCplx}}{\texttt{SumESFCplx}}$	$\frac{\texttt{DDSumESFCplx}}{\texttt{SumESFCplx}}$	$\frac{\texttt{CompSumESFCplx}}{\texttt{DDSumESFCplx}}$	$\frac{\texttt{CompSumESFCplx}}{\texttt{CompSumESFwErrCplx}}$
Env 1	5.42	13.30	41.19%	71.41%
Env 2	6.53	13.88	47.35%	78.89%
Env 3	4.93	11.36	43.44%	75.78%

6.3. *Simple application*

We consider symmetric indefinite matrices whose eigenvalues are well-conditioned, but the coefficients are ill-conditioned due to cancelation from subtractions. We choose the $n \times n$ matrices J and T presented in the section 5.3 and 5.4 of [2] as example, where

$$J = Q\begin{pmatrix} I_{n/2} & \\ & -I_{n/2} \end{pmatrix}Q^T$$

with Q be a random orthogonal matrix, and

$$T = \begin{pmatrix} 0 & 100 & & \\ 100 & \ddots & \ddots & \\ & \ddots & 0 & 100 \\ & & 100 & 0 \end{pmatrix}$$

be a tridiagonal Toeplitz matrix.

In the numerical tests, we let the tested matrix $A = J$ or T, $n = 100$, then every other coefficient is zero, that is $c_k = 0$ for k odd. Hence, we only consider the nozero coefficients computed here. We use the command `sym2poly(poly(vpa(A)))` in Symbolic Math Toolbox in Matlab to compute the exact coefficient c_k of the characteristic polynomial from the given matrix. We first compute the eigenvalues with the `eig` function in floating-point arithmetic and then determines the coefficients using `SumESF` or `CompSumESF`. We compare the accuracies of these two methods. The results are reported in Figs. 4 and 5.

From ϵ_{rel}, we deem that the computed eigenvalues have high relative accuracy. We observe that the relative perturbation $|\tilde{c}_k - c_k|/|c_k|$ denoted by the red real line is so much smaller than the perturbed bound in (44) when the condition numer is large. That is to say in these cases the perturbed bound is pessimistic. Hence, using higher precision in summation algorithm, such as `CompSumESF`, will enhance the accuracy of the coefficient and even keep it in the level of relative perturbation $|\tilde{c}_k - c_k|/|c_k|$, just shown in Figs. 4 and 5.

We find that for the matrix J, the eigenvalues are near the numbers 1 and -1; and for the matrix T, the eigenvalue $x_i = 2n\cos\frac{j\pi}{n+1}$, $a \le j \le n$. In contrast, we generate the eigenvalues by `GenCoef` with the condition number is 10^{15}. Then the eigenvalues will have enormous magnitudes. We let these eigenvalues have the perturbation with $\epsilon_{rel} = 10^{-14}$ and obtain the perturbed eigenvalue $\tilde{x}_i$. We observe that the relative perturbation $|\tilde{c}_k - c_k|/|c_k|$ is large up to 10^{-2}. In this case, the perturbed bound (44) is not pessimistic and Theorem 9 predicts the error bound well. Therefore we deem that the distribution of the eigenvalues can affect the relative perturbation.

The eigenvalues of the two tested matrices above are real numbers. Now we will focus on the Forsythe matrix, the eigenvalues of which are complex numbers. Then we can show our `CompSumESFCplx`'s effectiveness. We use F = gallery('forsythe',n,α, λ) in Matlab,with $n = 100$, $\alpha = \sqrt{eps}$ and $\lambda = 0$, to generate the following Forsythe matrix

$$F = \begin{pmatrix} 0 & 1 & & \\ & \ddots & \ddots & \\ & \ddots & 0 & 1 \\ \alpha & & & 0 \end{pmatrix}$$

The characteristic polynomial of F is $p(t) = t^n - \alpha$, which means that the coefficients are zero except $c_{100} = -\alpha$. Then we compare the absolute errors of `CompSumESFCplx` and `SumESFCplx`. Just like the experiments before, We first compute the eigenvalues

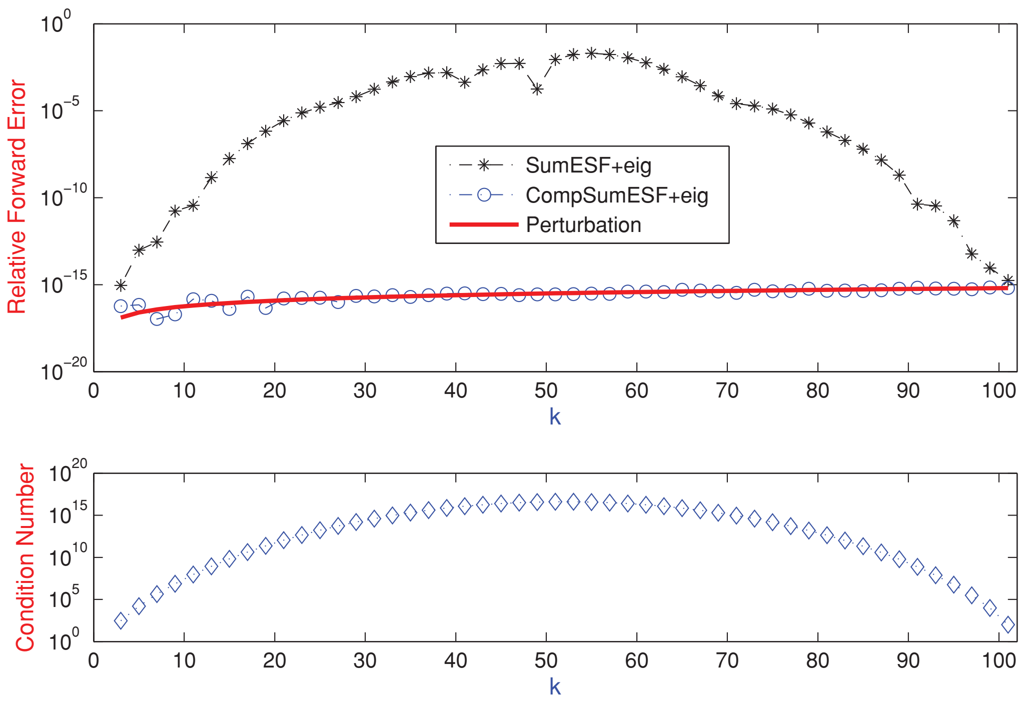

Fig. 4. Symmetric indefinite matrix J, with $\epsilon_{rel} \approx 2.0 \times 10^{-15}$.

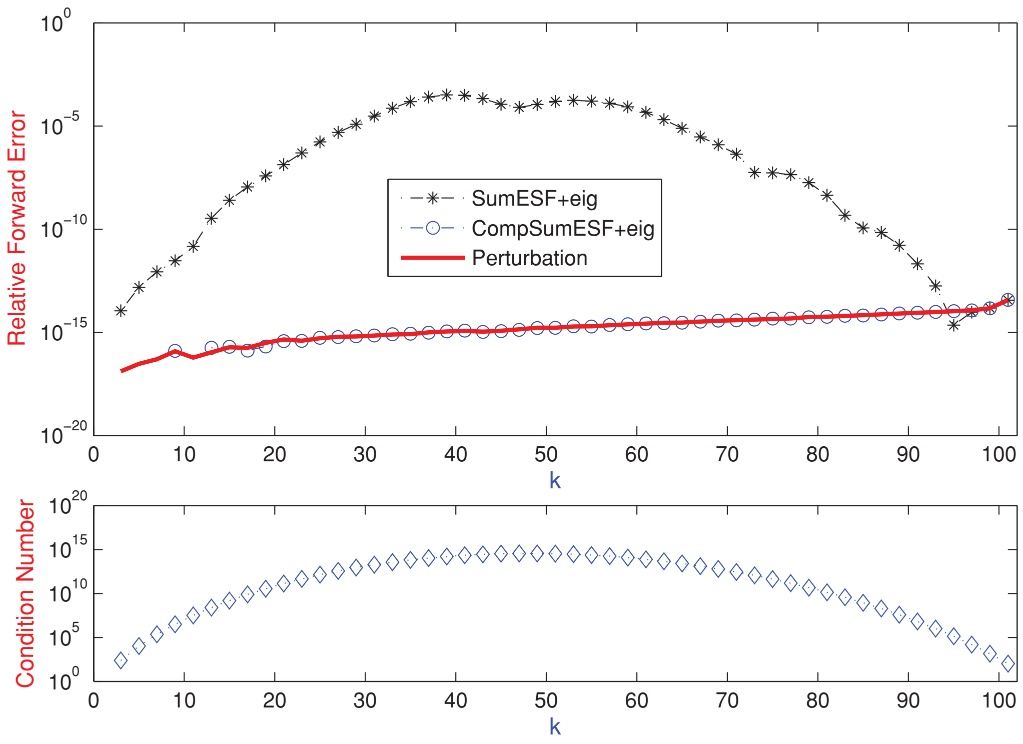

Fig. 5. Symmetric indefinite tridiagonal Toeplitz matrix T with $\epsilon_{rel} \approx 2.0 \times 10^{-14}$.

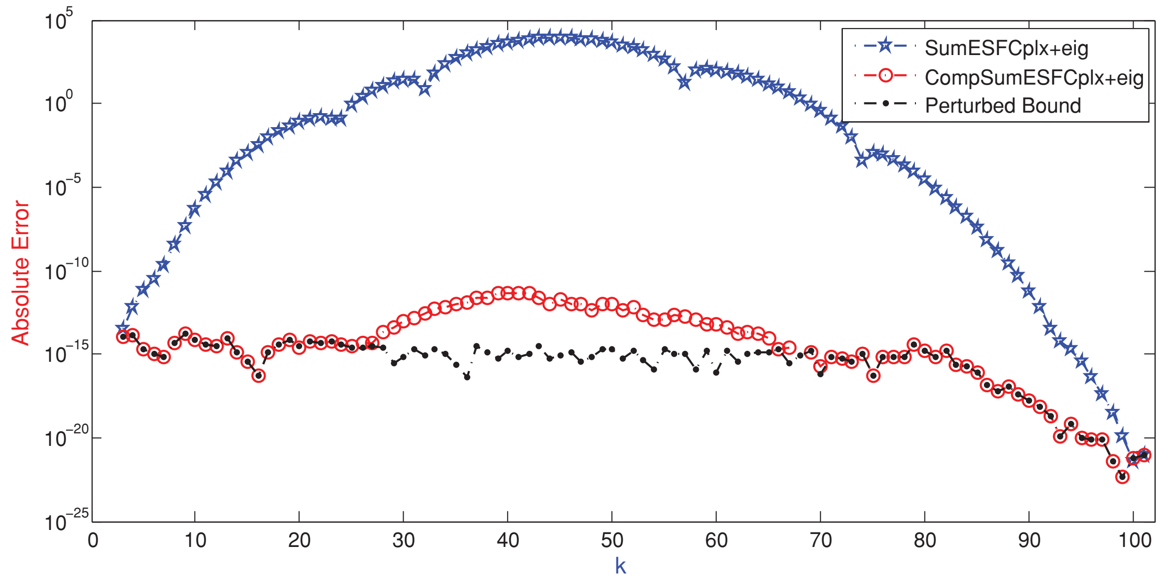

Fig. 6. The Forsythe matrix F with $\epsilon_{abs} \approx 1.67$.

with the `eig` function in floating-point arithmetic, the largest absolute error of which is $\epsilon_{abs} \approx 1.67$. Then we determine the coefficients using `SumESFCplx` and `CompSumESFCplx` from these eigenvalues. From Fig. 6, it can be observed that many coefficients computed by `SumESFCplx` are not only nonzero but have enormous magnitudes. However, `CompSumESFCplx` can give the similar absolute errors as the `plot` function using symbolic method. It is interesting that `SumESFCplx` can also give the high accurate results if we reorder the eigenvalues with the `sort` instruction in Matlab. We deem that the complex eigenvalues will introduce more roundoff errors when they are in disorder.

Whatever, `CompSumESF` and `CompSumESFCplx` show their effectiveness on computing characteristic polynomial from matrix in some cases.

7. Conclusions

We have presented two compensated summation algorithms for the computation of the elementary symmetric functions with the real and complex floating-point inputs, respectively. The algorithms yield results as accurate as if computed by the traditional algorithm in twice the working precision but using standard double precision. We compared these algorithms with the summation algorithm in double–double format and showed that our algorithms were faster while sharing the same accuracy. These algorithms also performed well on the application of computing the coefficients of polynomials from zeros and computing the characteristic polynomial from matrix.

References

[1] G. Fischer, Einführung in die Theorie psychologischer tests: Grundlagen und Anwendungen, Huber, Bern, Switzerland, 1974.
[2] R. Rehman, I. Ipsen, Computing characteristic polynomials from eigenvalues, SIAM J. Matrix. Anal. Applicat. 32 (1) (2011) 90–114.
[3] S. Graillat, Accurate floating-point product and exponentiation, IEEE Trans. Comput. 58 (7) (2009) 994–1000.
[4] S. Graillat, P. Langlois, N. Louvet, Algorithms for accurate, validated and fast polynomial evaluation, Jpn. J. Ind. Appl. Math. 26 (2) (2009) 215–231.
[5] S. Graillat, V. Morain, Accurate summation, dot product and polynomial evaluation in complex floating-point arithmetic, Inf. Comput. 216 (2012) 57–71.
[6] P. Kornerup, C. Lauter, V. Lefèvre, N. Louvet, J. Muller, Computing correctly rounded integer powers in floating-point arithmetic, ACM Trans. Math. Software 37 (1) (2010) 4:1–4:23.
[7] T. Ogita, S.M. Rump, S. Oishi, Accurate sum and dot product, SIAM J. Sci. Comput. 26 (6) (2005) 1955–1988.
[8] S.M. Rump, Ultimately fast accurate summation, SIAM J. Sci. Comput. 31 (5) (2009) 3466–3502.
[9] S.M. Rump, T. Ogita, S. Oishi, Accurate floating-point summation part I: Faithful rounding, SIAM J. Sci. Comput. 31 (1) (2006) 189–224.
[10] S.M. Rump, T. Ogita, S. Oishi, Accurate floating-point summation part II: Sign, *k*-fold faithful and rounding to nearest, SIAM J. Sci. Comput. 31 (2) (2008) 1269–1302.
[11] D. Calvetti, L. Reichel, On the evaluation of polynomial coefficients, Numer. Algorithms 33 (2003) 153–161.
[12] A. Eisinberg, G. Fedele, A property of the elementary symmetric functions, Calcolo 42 (1) (2005) 31–36.
[13] F. Baker, M. Harwell, Computing elementary symmetric functions and their derivatives: A didactic, Appl. Psychol. Meas. 20 (2) (1996) 169–192.
[14] H. Jiang, S. Graillat, R. Barrio, Accurate and Fast Evaluation of Elementary Symmetric Functions, in: Proceedings of 21st IEEE Symposium on Computer Arithmetic, IEEE Computer Society, 2013, pp. 183–190.
[15] N. Higham, Accuracy and Stability of Numerical Algorithms, second ed., SIAM, Philadelphia, 2002.
[16] D. Knuth, The art of computer programming: Seminumerical algorithms, 2, third ed., Addison-Wesley, 1998.
[17] T.J. Dekker, A floating-point technique for extending the available precision, Numer. Math 18 (3) (1971) 224–242.
[18] P. Langlois, N. Louvet, How to ensure a faithful polynomial evaluation with the compensated Horner algorithm, in: Proceedings of 18th IEEE Symposium on Computer Arithmetic, IEEE Computer Society, 2007, pp. 141–149.

[19] S. Graillat, V. Morain, Error-free transformations in real and complex floating-point arithmetic, in: Proceedings of the International Symposium on Nonlinear Theory and its Applications, 2007, pp. 341–344.
[20] O. Caprani, Roundoff errors in floating-point summation, BIT 15 (1975) 5–9.
[21] H. Jiang, S. Graillat, C.B. Hu, S.G. Li, X.K. Liao, L.Z. Cheng, F. Su, Accurate evaluation of the kth derivative of a polynomial and its application, J. Comput. Appl. Math. 243 (2013) 28–47.
[22] W. Miller, Graph transformations for roundoff analysis, SIAM J. Comput. 5 (1976) 204–216.
[23] R. Rehman, Numerical computation of the characteristic polynomial of a complex matrix, North Carolina State University, Raleigh, NC, 2010 Phd thesis.
[24] D.H. Bailey, Y. Hida, X.S. Li, B. Thompson, ARPREC: an arbitrary precision computation package, Technical report, Lawrence Berkeley National Laboratory, 2002.
[25] R.P. Brent, A FORTRAN multiple-precision arithmetic package, ACM Trans. Math. Softw. 4 (1) (1978) 57–70.
[26] L. Fousse, G. Hanrot, V. lefèvre, P. Pélissier, P. Zimmermann, MPFR: A Multiple-precision binary floating-point library with correct rounding, ACM Trans. Math. Softw. 33 (2) (2007) 13:1–13:15.
[27] D.H. Bailey, Library for Double-Double and Quad-Double Arithmetic (QD library), Retrieved from http://crd-legacy.lbl.gov/~dhbailey/mpdist/, (accessed 26.08.15).
[28] X. Li, J. Demmel, D. Bailey, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, M. Martin, et al., Design, implementation and testing of extended and mixed precision blas, ACM Trans. Math. Softw. 28 (2) (2002) 152–205.
[29] N. Louvet, Compensated algorithms in floating-point arithmetic: accuracy, validation, performances, Université de Perpignan Via Domitia, 2007 Phd thesis.
[30] C. Lauter, Basic building blocks for a triple-double intermediate format, Technical report RR2005-38, LIP, France, 2005.
[31] Y. Hida, X. Li, D.H. Bailey, Algorithms for quad-double precision floating-point arithmetic, in: Proceedings 15th IEEE Symposium on Computer Arithmetic, IEEE Computer Society, 2001, pp. 155–162.
[32] P. Langlois, N. Louvet, More instruction level parallelism explains the actual efficiency of compensated algorithms, Technical report, hal-00165020, DALI Research Team, University of Perpignan, France, 2007.