2013 IEEE 21st Symposium on Computer Arithmetic

Accurate and Fast Evaluation of Elementary Symmetric Functions

Hao Jiang*, Stef Graillat† and Roberto Barrio‡
*School of Science, National University of Defense Technology, Changsha, China
Email: jhnudt@yahoo.cn
†PEQUAN, LIP6, Université Pierre et Marie Curie, CNRS, Paris, France
Email: stef.graillat@upmc.fr
‡Dpto. de Matemática Aplicada and IUMA, Universidad de Zaragoza, E-50009 Zaragoza, Spain
Email: rbarrio@unizar.es

***Abstract*—This paper is concerned with the fast and accurate evaluation of elementary symmetric functions. We present a new compensated algorithm by applying error-free transformations to improve the accuracy of the so-called Summation Algorithm, which is used, by example, in the MATLAB's `poly` function. We derive a forward roundoff error bound and running error bound for our new algorithm. The roundoff error bound implies that the computed result is 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 algorithm runs much faster than the algorithm using the classic double-double library while sharing similar error estimates. Such an algorithm can be widely applicable for example to compute characteristic polynomials from eigenvalues. It can also be used into the Rasch model in psychological measurement.**

***Keywords*-elementary symmetric functions; floating-point arithmetic; roundoff error; error-free transformation; compensated algorithm; accurate algorithm.**

I. Introduction

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

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

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

The classic and widely-used method to compute the elementary symmetric function (1) is the so-called Summation Algorithm [9], which is essentially the algorithm used by MATLAB's `poly` function. The error analysis of this algorithm has been considered in [26], and the result implies the algorithm is stable. However, as mentioned in [26] "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 [11], [12], [19], [25], [27], [28], [29], we propose a fast and accurate compensated algorithm, 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 can be found in [25] and [11], respectively.

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

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

With the Summation Algorithm, one can evaluate polynomial's coefficients $\{c_i\}_{i=0}^n$ from its zeros $\{x_i\}_{i=1}^n$, specially compute characteristic polynomials from eigenvalues (see [5], [8] and [26]). Our algorithm 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 [3]. 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 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, numerical experiments illustrate the accuracy and efficiency of our compensated algorithm. Finally, concluding remarks and future work are left for Section 5.

This paper is a full version of the poster [16] that only presented the compensated algorithm and its properties. Here we perform the detailed error analysis and numerical tests.

1063-6889/13 $26.00 © 2013 IEEE
DOI 10.1109/ARITH.2013.18

IEEE computer society

II. Notations and Preliminaries

A. 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

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

where op $\in \{\oplus, \ominus, \otimes\}$, $\circ \in \{+,-,\times\}$ and $|\varepsilon_1|, |\varepsilon_2| \leq u$ 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 $\mathrm{fl}(a)$ 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 $|\delta_i| \leq u$, $\rho_i = \pm 1$, then $\prod_{i=1}^{n}(1+\delta_i)^{\rho_i} = 1 + \theta_n$,
2) $1 + \theta_n = \langle n \rangle$ and $|\theta_n| \leq \gamma_n := nu/(1-nu)$,
3) $(1+\theta_k)(1+\theta_j) = (1+\theta_{k+j})$, $\langle k \rangle \langle j \rangle = \langle k+j \rangle$,
4) $\gamma_k + \gamma_j + \gamma_k\gamma_j \leq \gamma_{k+j}$ and $\gamma_k < \gamma_{k+1}$.

To derive the running error bound, we need the next relations obtained from [12] and [20].

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

B. 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 `TwoProductFMA`. We can see the details of the four algorithms above in the references presented in Table I. 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.[25]) For $a, b, x, y \in \mathbb{F}$, $[x,y] = \mathtt{TwoSum}(a,b)$ verifies

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

Table I
Error-Free Transformations of Addition and Product

Algorithm	Flops	Ref.
$[x,y] = \mathtt{TwoSum}(a,b)$	6	[18]
$[x,y] = \mathtt{Split}(a)$	4	[7]
$[x,y] = \mathtt{TwoProd}(a,b)$	17	[7]
$[x,y] = \mathtt{FastTwoSum}(a,b)$	3	[7]
$[x,y] = \mathtt{TwoProductFMA}(a,b)$	2	[25]

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

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

C. 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 *k*th ESF evaluation (1) as

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

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

$$\mathtt{cond}(S_k^{(n)}(X)) = \frac{k S_k^{(n)}(|X|)}{|S_k^{(n)}(X)|}. \quad (5)$$

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

D. Classic Algorithm

The Summation Algorithm, represented by Algorithm 1 below, computes the elementary symmetric functions recursively, which is the same as the one in [26], except that it only computes the *k*th ESF rather than all of ESFs.

```
Algorithm 1: Summation Algorithm
Input: $X = (x_1, \ldots, x_n)$ and $k$
Output: k-th ESF $S_k^{(n)}(X) = S_k^{(n)}$
function $S_k^{(n)}$=SumESF($X, k$)
$S_0^{(i)} = 1,\ 1 \leq i \leq n-1;\ S_j^{(i)} = 0,\ j > i;\ S_1^{(1)} = x_1;$
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
```

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

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

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 *k*th ESF. However, in practical calculation such as computing characteristic polynomial from eigenvalue, this substitution is often required.

The following theorem gives roundoff error bounds for Algorithm 1.

Theorem 2: If $X = (x_1, \ldots, x_n)$ is a vector of floating-point numbers, the computed k-th elementary symmetric function $\widehat{S}_k^{(n)} = \widehat{S}_k^{(n)}(X)$ by Algorithm 1 in floating-point arithmetic verifies

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

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 $2 \le k \le n-1$, one method is the induction shown in [26]. However we deem that the error bound of $\theta_t^{(i_1 \ldots i_k)}$ in Theorem 4.3 of [26] should be $\gamma_{2(n-1)}$. Hence, we make a small improvement by substituting $\gamma_{2(n-1)}$ for γ_{2n}. The other method is using data dependency graph just like that in [6], [17] and [30]. Then it is easy to obtain the following equation

$$|\widehat{S}_j^{(i)} - S_j^{(i)}| \le \gamma_{2(i-1)} S_j^{(i)}(|x_1|, \ldots, |x_i|), 1 \le j \le i \le n. \tag{6}$$

Let $j = k$ and $i = n$, we will obtain the expected result. Finally by definition of the condition number (5), we can obtain the relative error bound of Algorithm 1. ■

III. Accurate ESF evaluation

In this section, we present a compensated algorithm to compute the ESFs based on error-free transformations and classic Summation Algorithm. The result 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 3).

A. Compensated Algorithm

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

```
Algorithm 2: Compensated Summation Algorithm
Input: $X = (x_1, \ldots, x_n)$ 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

$$p + \beta_j^{(i)} = x_i \times \widehat{S}_{j-1}^{(i-1)} \text{ and } \widehat{S}_j^{(i)} + \sigma_j^{(i)} = \widehat{S}_j^{(i-1)} + p, \tag{7}$$

and then

$$\widehat{S}_j^{(i)} + (\beta_j^{(i)} + \sigma_j^{(i)}) = \widehat{S}_j^{(i-1)} + x_i \times \widehat{S}_{j-1}^{(i-1)}. \tag{8}$$

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

$$\widehat{S}_j^{(i)} + \epsilon S_j^{(i)} = S_j^{(i)},\ \forall\ i, j. \tag{9}$$

Since

$$S_j^{(i)} = S_j^{(i-1)} + x_i \times S_{j-1}^{(i-1)}, \tag{10}$$

and by (8), (9) and (10), we can deduce that

$$\epsilon S_j^{(i)} = \epsilon S_j^{(i-1)} + (\beta_j^{(i)} + \sigma_j^{(i)}) + x_i \times \epsilon S_{j-1}^{(i-1)}. \tag{11}$$

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.

B. Forward Error Bound

Lemma 1: Let us consider the recurrence (11), with the notation $w_j^{(i)} = \beta_j^{(i)} + \sigma_j^{(i)}$ and Definition (1). Then, we have

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

Proof: The proof is direct by induction. We can also obtain the result by drawing data dependency graph like in [6], [17] and [30]. ■

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

$$\begin{aligned}\widehat{\epsilon S}_j^{(i)} &= \widehat{\epsilon S}_j^{(i-1)} \oplus \widehat{w}_j^{(i)} \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)} \\ &= \widehat{w}_j^{(i)}\langle 2\rangle + \widehat{\epsilon S}_j^{(i-1)}\langle 2\rangle + x_i \times \widehat{\epsilon S}_{j-1}^{(i-1)}\langle 2\rangle.\end{aligned}$$

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

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

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

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

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

$$\begin{aligned}|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \gamma_{2(n-1)} \sum_{i=2}^{n} \sum_{j=max\{1,i+k-n\}}^{min\{i,k\}} |w_j^{(i)}| \\ \times S_{k-j}^{(n-i)}(|x_{i+1}|,\dots,|x_n|).\end{aligned} \tag{12}$$

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

$$|w_j^{(i)}| \le |\beta_j^{(i)}| + |\sigma_j^{(i)}| \le 2u(1+u)(|\widehat{S}_j^{(i-1)}| + |x_i| \times |\widehat{S}_{j-1}^{(i-1)}|). \tag{13}$$

Taking into account (6), we have

$$\begin{aligned}|\widehat{S}_j^{(i-1)}| &\le (1+\gamma_{2(i-2)})S_j^{(i-1)}(|x_1|,\dots,|x_{i-1}|), \\ |\widehat{S}_{j-1}^{(i-1)}| &\le (1+\gamma_{2(i-2)}) \times S_{j-1}^{(i-1)}(|x_1|,\dots,|x_{i-1}|).\end{aligned} \tag{14}$$

Then, considering (13), (14) and

$$\begin{aligned}& S_j^{(i-1)}(|x_1|,\dots,|x_{i-1}|) + |x_i| \times S_{j-1}^{(i-1)}(|x_1|,\dots,|x_{i-1}|) \\ &= S_j^{(i)}(|x_1|,\dots,|x_i|),\end{aligned}$$

we obtain

$$|w_j^{(i)}| \le 2u(1+u)(1+\gamma_{2(n-2)})S_j^{(i)}(|x_1|,\dots,|x_i|). \tag{15}$$

Since

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

then from (12), (15), we can finally deduce the expected result. ■

Theorem 3: For a vector of n floating-point numbers $X = (x_1,\dots,x_n)$, the relative forward error bound in Algorithm 2 satisfies

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

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

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

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

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le u|S_k^{(n)}| + \gamma_{2(n-1)}^2 S_k^{(n)}(|X|). \tag{16}$$

At last the desired bound follows from the definition of condition number (5). For the cases of $k=1$ and $k=n$, the results come from Proposition 4.5 of [25] and Theorem 2 of [11], respectively. ■

C. Running Error Analysis

In practical calculations, it is desirable to obtain a corresponding error bound at the same time as the computed value. The *a priori* error bound (16) of Theorem 3 is entirely adequate for theoretical purposes, but lakes sharpness. For this requirement, we perform a running error analysis of the `CompSumESF` algorithm, 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 4.

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 $j = \max\{1, i+k-n\} : \min\{i, k\}$
    $[p, \beta_j^{(i)}] = \texttt{TwoProd}(x_i, \widehat{S}_{j-1}^{(i-1)})$;
    $[\widehat{S}_j^{(i)}, \sigma_j^{(i)}] = \texttt{TwoSum}(\widehat{S}_j^{(i-1)}, p)$;
    $\widehat{\epsilon S}_j^{(i)} = \widehat{\epsilon S}_j^{(i-1)} \oplus (\beta_j^{(i)} \oplus \sigma_j^{(i)}) \oplus x_i \otimes \widehat{\epsilon S}_{j-1}^{(i-1)}$
    $\widehat{ES}_j^{(i)} = \widehat{ES}_j^{(i-1)} \oplus |\beta_j^{(i)} \oplus \sigma_j^{(i)}| \oplus |x_i| \otimes \widehat{ES}_{j-1}^{(i-1)}$
  end
end
$[\overline{S}_k^{(n)}, c] = \texttt{FastTwoSum}(\widehat{S}_k^{(n)}, \widehat{\epsilon S}_k^{(n)})$
$\hat{\alpha} = (\widehat{\gamma}_{2(n-1)} \otimes \widehat{ES}_k^{(n)}) \oslash (1 - 3nu)$
$\mu = (|c| \oplus \hat{\alpha}) \oslash (1 - 2u)$
```

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

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le fl\left(\frac{\widehat{\gamma}_{2(n-1)} \otimes \widehat{ES}_k^{(n)}}{1 - 3nu}\right) := \hat{\alpha} \qquad (17)$$

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

$$\widehat{ES}_j^{(i)} = \widehat{ES}_j^{(i-1)} \oplus |\beta_j^{(i)} \oplus \sigma_j^{(i)}| \oplus |x_i| \otimes \widehat{ES}_{j-1}^{(i-1)} \qquad (18)$$

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

Proof: From (12) in Lemma 2, we have

$$|\epsilon S_k^{(n)} - \widehat{\epsilon S}_k^{(n)}| \le \gamma_{2(n-1)} ES_k^{(n)}, \qquad (19)$$

with

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

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 (18) 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 $t = 1, \dots, 4$. 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

$$(1+u)^{3(n-1)} \widehat{ES}_k^{(n)} \ge ES_k^{(n)}. \qquad (20)$$

Table II
ADDITION AND MULTIPLICATION IN DOUBLE-DOUBLE FORMAT

Algorithm	Flops	Ref.
$[rh, rl] = \texttt{add\_dd\_dd}(ah, al, bh, bl)$	20	[23]
$[rh, rl] = \texttt{prod\_dd\_d}(ah, al, b)$	22	[23], [24]

Finally, by the first part in (4) and (20), it follows from (19) 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 (17). ■

Theorem 4: Assume $3nu < 1$, then a running error bound of Algorithm 2 is given by

$$|\overline{S}_k^{(n)} - S_k^{(n)}| \le fl\left(\frac{|c| \oplus \widehat{\alpha}}{1 - 2u}\right) := \mu, \qquad (21)$$

where $\widehat{\alpha}$ is the error bound defined by (17) 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

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

■

In order to obtain a running relative error bound, when $\overline{S}_k^{(n)} \neq 0$, we may deduce by (21) that

$$\begin{aligned} \frac{|\overline{S}_k^{(n)} - S_k^{(n)}|}{|S_k^{(n)}|} &\le \frac{\mu}{|S_k^{(n)}|} \le \frac{\mu}{|\overline{S}_k^{(n)}(1 - \frac{\mu}{\overline{S}_k^{(n)}})|} \\ &\le \frac{\mu}{|\overline{S}_k^{(n)}|} + \mathcal{O}\left(\left(\frac{\mu}{|\overline{S}_k^{(n)}|}\right)^2\right). \end{aligned} \qquad (22)$$

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 along with the computed result, without requiring the additional significant computational cost.

D. 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 [2], MP [4], and MPFR [10], but if we just want to double the IEEE-754 double precision, a more efficient way is to use Bailey's double-double arithmetic [1]. A double-double number a is the pair (ah, al) of floating-point numbers with $a = ah + al$, $|al| < u|ah|$ and $|al| < u|a|$.

In the sequel, we present two algorithms to compute the addition of two double-double and a double times a double-double shown in Table II.

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

```
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] = $ prod_dd_d$(Sh_{j-1}^{(i-1)}, Sl_{j-1}^{(i-1)}, x_i)$;
    $[Sh_j^{(i)}, Sl_j^{(i)}] = $ add_dd_dd$(rh, rl, Sh_j^{(i-1)}, Sl_j^{(i-1)})$
  end
end
```

Before studying the accuracy of the result of Algorithm 4, let us see the model of floating-point double-double arithmetic, the similar results and proof can be seen in [22].

***Remark* 1:** *([23, p.182])* For a standard model of floating-point arithmetic for the double-double algorithms

$$fl(a \odot b) = (a \odot b)(1 + \delta), \tag{23}$$

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

$$|\delta| \le u_{dd}, for \odot \in \{+, -\};\ |\delta| \le 2u_{dd}, for \odot \in \{\times, /\} \tag{24}$$

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

Theorem 5: 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

$$\overline{\gamma}_{3(n-1)} = \frac{3(n-1)u_{dd}}{1 - 3(n-1)u_{dd}} = \frac{6(n-1)u^2}{1 - 6(n-1)u^2}.$$

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. ■

IV. 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: `SumESF`, `CompSumESF`, `CompSumESFwErr` and `DDSumESF`. All accuracy measurements are done in MATLAB 7.0. All timing tests are done on a personal laptop with Intel Pentium Dual CPU E2160, each at 1.8 GHz, and with Microsoft Visual C++ 2008 with the default compiler option /od on Windows 7.

A. Accuracy test

To test accuracy of our algorithms we need extremely ill-conditioned elementary symmetric functions, hence we designed Algorithm 5 – `GenESF`, shown in Appendix A, based on `GenPoly` [24] and `GenDot` [25]. The basic idea of this generation algorithm is constructing random vector X with $S_k^{(n)}(|X|) \approx$ cond_exp $\times\, |v|$ and $S_k^{(n)}(X) \approx v$. We generated the floating-point vector[1] $X \in \mathbb{F}^n$ 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 *k*th 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 [14], which is also an option of function `ExactESF` in Appendix A, besides of Symbolic method. We also assume that the relative errors greater than 2 be the value 2 just like [25], which means there is no useful information left.

As we can see on Figure 1, `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 Figure 1. It is also shown that the forward error bound from Theorem 3 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 [5]). 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.

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

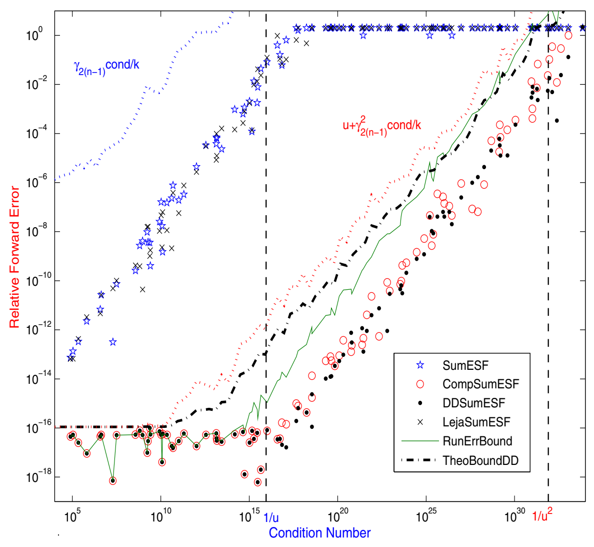

Figure 1. Accuracy of evaluation with respect to the condition number.

B. Running time test

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 [5] and be used in MATLAB's `poly`. There are some other optimizations just like that in [24], 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 $[-1, 1]$ with n varying from 10 to 30.

In this part, we perform 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 $j = 1 : i$. We exhibit the measured running time ratios in two cases in Table III. 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.

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 approximatly 11.5, which is much smaller than

Table III
TIME RATIOS OF COMPUTING k-TH ESF AND ALL ESFS

	$\frac{\texttt{CompSumESF}}{\texttt{SumESF}}$	$\frac{\texttt{DDSumESF}}{\texttt{SumESF}}$	$\frac{\texttt{CompSumESF}}{\texttt{DDSumESF}}$	$\frac{\texttt{CompSumESF}}{\texttt{CompSumESFwErr}}$
Case 1	3.05	5.42	57.42%	69.91%
Case 2	3.91	7.48	52.97%	68.02%

the running time ratios 7.48 shown in Table III. Thanks to the analysis in terms of instruction level parallelism (ILP) (see details in [21], [24]), 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.

V. Conclusions and Future Work

We have presented a compensated Summation Algorithm for the computation of the elementary symmetric functions with the real floating-point inputs. This algorithm yields a result as accurate as if computed by the traditional algorithm in twice the working precision but using standard double precision. We compared this algorithm with the Summation Algorithm in double-double format and showed that our algorithm was faster while sharing the same accuracy. This algorithm also performed well on the application of computing the coefficients of polynomials from zeros.

In this paper, we have only dealt with the case of real floating-point inputs, we will consider designing the corresponding compensated Summation Algorithm for complex floating-point inputs based on the results of [13]. The computation of ESF's derivative is also an important part of CMLE in the Rash model, we will propose the corresponding fast and accurate algorithm. And it is foreseeable that even if underflow occurs, our algorithm would still show excellent behavior, only requiring a more sophisticated error analysis. All these will be considered in the future.

References

[1] D.H. Bailey, "Library for Double-Double and Quad-Double Arithmetic (QD library)," Available from < http://www.nersc.gov/∼dhbailey/mpdist/mpdist.html>.

[2] D.H Bailey, Y. Hida, X.S. Li, and B. Thompson, "ARPREC: an arbitrary precision computation package," *Technical report* Lawrence Berkeley National Laboratory, 2002.

[3] F. Baker and M. Harwell, "Computing elementary symmetric functions and their derivatives: A didactic," *Applied Psychological Measurement*, vol. 20, no. 2, pp. 169-192, 1996.

[4] R.P. Brent, "A FORTRAN multiple-precision arithmetic package," *ACM Transactions Mathematical Software*, vol. 4, no. 1, pp. 57-70, 1978.

[5] D. Calvetti and L. Reichel, "On the evaluation of polynomial coefficients. *Numerical Algorithms*, vol.33, pp.153-161, 2003.

[6] O. Caprani, "Roundoff Errors in Floating-Point Summation," *BIT Numerical Mathematics.*, vol. 15, pp.5-9, 1975.

[7] T.J. Dekker, " A floating-point technique for extending the available precision," *Numerische Mathematik,* vol. 18, no. 3, pp. 224-242, 1971.

[8] A. Eisinberg and G. Fedele, "A property of the elementary symmetric functions," *Calcolo*, vol. 42, no. 1, pp. 31–36, 2005.

[9] G. Fischer, Einführung in die Theorie psychologischer tests: Grundlagen und Anwendungen, Huber, Bern, Switzerland, 1974.

[10] L. Fousse, G. Hanrot, V. lefèvre, P. pélissier, and P. Zimmermann, "MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding," *ACM Transactions Mathematical Software*, vol. 33, no. 2, pp.13:1-13:15, 2007.

[11] S. Graillat, "Accurate floating-point product and exponentiation," *IEEE Transactions on Computers,* vol. 58, no. 7, pp. 994-1000, 2009.

[12] S. Graillat, P. Langlois, and N. Louvet, "Algorithms for accurate, validated and fast polynomial evaluation," *Japan Journal of Industrial and Applied Mathematics,* vol. 26, no. 2, pp. 215-231, 2009.

[13] S. Graillat and V. Morain, "Error-free transformations in real and complex floating point arithmetic,"In *Proceedings of the International Symposium on Nonlinear Theory and its Applications*, Vancouver, Canada, Sept. 2007, pp. 341-344.

[14] Y. Hida, X. Li, and D. Bailey, "Algorithms for quad-double precision floating point arithmetic. In *Proceedings 15th IEEE Symposium on Computer Arithmetic*, N. Burgess and L. Ciminiera, Eds., Vail, CO, Jun. 2001, pp. 155-162

[15] N. Higham, Accuracy and stability of numerical algorithms, SIAM, Philadelphia, second edition, 2002.

[16] H. Jiang, S. Graillat, R. Barrio, "Accurate computing elementary symmetric functions," *ACM Communications in Computer Algebra* (ISSAC'12 poster), vol. 46, no. 3, pp.102-103, 2012.

[17] H. Jiang, S. Graillat, C.B. Hu, S.G. Li, X.K. Liao, L.Z. Cheng, F. Su, "Accurate evaluation of the k-th derivative of a polynomial and its application," *Journal of Computational and Applied Mathematics*, vol. 243, pp.28-47, 2013.

[18] D. Knuth, The art of computer programming: Seminumerical algorithms, volume 2, Addison-Wesley, third edition, 1998.

[19] P. Kornerup, C. Lauter, V. Lefèvre, N. Louvet, and J. Muller, " Computing correctly rounded integer powers in floating-point arithmetic," *ACM Transactions Mathematical Software*, vol. 37, no. 1, pp.4:1-4:23, 2010.

[20] P. Langlois and N. Louvet, "How to ensure a faithful polynomial evaluation with the compensated Horner algorithm," In *Proceedings 18th IEEE Symposium on Computer Arithmetic*, Montpellier, France, Jun. 2007, pp. 141-149.

[21] P. Langlois and 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.

[22] C. Lauter, "Basic building blocks for a triple-double intermediate format," Technical report RR2005-38, LIP, France, 2005.

[23] 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 Transactions Mathematical Software*, vol. 28, no. 2, pp. 152–205, 2002.

[24] N. Louvet, "Compensated algorithms in floating point arithmetic: accuracy, validation, performances," PhD thesis, Université de Perpignan Via Domitia, 2007.

[25] T. Ogita, S. Rump, and S. Oishi, "Accurate sum and dot product," *SIAM Journal on Scientific Computing*, vol. 26, no. 6, pp. 1955–1988, 2005.

[26] R. Rehman and I. Ipsen, "Computing characteristic polynomials from eigenvalues," *SIAM Journal on Matrix Analysis and Applications*, vol. 32, no. 1, pp. 90–114, 2011.

[27] S. Rump, "Ultimately fast accurate summation. *SIAM Journal on Scientific Computing*, vol. 31, no. 5, pp. 3466-3502, 2009.

[28] S. Rump, T. Ogita, and S. Oishi, "Accurate floating-point summation part I: Faithful rounding. *SIAM Journal on Scientific Computing*, vol. 31, no. 1, pp. 189-224, 2008.

[29] S. Rump, T. Ogita, and S. Oishi, "Accurate floating-point summation part II: Sign, k-fold faithful and rounding to nearest," *SIAM Journal on Scientific Computing*, vol. 31, no. 2, pp. 1269-1302, 2008.

[30] W. Miller, "Graph Transformations for Roundoff Analysis," *SIAM Journal on Computing*, vol. 5, pp. 204-216, 1976.

APPENDIX

Algorithm 5: The generation algorithm of ill-conditioned elementary symmetric function

```
function $[x, cond\_real] = \mathtt{GenESF}(n, cond\_exp, coef\_num)$
% input:   n          the dimension
%          cond_exp   the expected condition number
%          coef_num   k-th elementary symmetric function's 'k'
% output:  x          the real vector
%          cond_real  the actual condition number
% uses: ExactESF(x,k) calculating exact result $S_k^{(n)}(X)$
  $n2 = \mathtt{round}(n/2)$;
  $x = \mathtt{zeros}(n, 1)$;
  $b = \mathtt{log2}(cond\_exp)/coef\_num$;
  $e = \mathtt{round}(\mathtt{rand}(n2, 1) * b)$;
  $e(1 : coef\_num) = (b + 1). * \mathtt{ones}(coef\_num, 1)$;
  if $coef\_num > n2$
    $n2 = coef\_num$;
  end
  $prod = 1$;
  for $i = 1 : coef\_num$
    $prod = prod * x(i)$;
  end
  $imp =$
  $\mathtt{power}(cond\_exp/\mathtt{abs}(prod * coef\_num), 1/coef\_num)$;
  $x(1 : coef\_num) = x(1 : coef\_num). * imp$;
  $e = \mathtt{round}(\mathtt{linspace}(b, 0, n - n2))$;
  for $i = n2 + 1 : n$
    $t1 = \mathtt{ExactESF}(x(1 : i - 1), coef\_num)$;
    $t2 = \mathtt{ExactESF}(x(1 : i - 1), coef\_num - 1)$;
    $x(i) = ((2 * \mathtt{rand} - 1) * 2^e(i - n2) - t1)/t2$;
  end
  $index = \mathtt{randperm}(n)$;
  $x = x(index)$;
  $d = \mathtt{ExactESF}(x, coef\_num)$;
  $D = \mathtt{ExactESF}(\mathtt{abs}(x), coef\_num)$;
  $cond\_real = coef\_num * D/\mathtt{abs}(d)$;
```