Verified Numerical Computations for Searching for Vectors with the Maximum Sum

Yuki Uchino¹,* and Katsuhisa Ozaki²

¹Graduate School of Engineering and Science, Shibaura Institute of Technology
²Department of Mathematical Sciences, Shibaura Institute of Technology

*mf20013@shibaura-it.ac.jp

Received: December 8, 2020; Accepted: February 18, 2021; Published: March 8, 2021

Abstract. This study proposes a numerical method for searching for vectors whose sum of all elements is maximum. The vector search can be applied to many scientific problems. If we use numerical computations without considering rounding errors, in a worst-case scenario, we might find incorrect vectors as a result. We propose herein an adaptive method designed to work accurately, regardless of the rounding error problems, based on an error-free transformation of a floating-point vector.

Keywords: floating-point arithmetic, verified numerical computations, accurate numerical algorithms

1. Introduction

This study focuses on the sum of all the elements of a floating-point vector. Let \( \mathbb{F} \) be a set of fixed precision binary floating-point numbers defined by IEEE 754 [1]. Let \( m_i \in \mathbb{N} \) and \( p^{(i)} = (p_1^{(i)}, p_2^{(i)}, \ldots, p_{m_i}^{(i)})^T \in \mathbb{R}^{m_i} \), for \( i = 1, \ldots, n \). We propose herein a numerical method finding the following set:

\[
I := \left\{ j \mid \sum_{k=1}^{m_j} p_k^{(j)} = \max_{1 \leq i \leq n} \left( \sum_{k=1}^{m_i} p_k^{(i)} \right) \right\}. \tag{1}
\]

We call vectors \( p^{(i)} \) for \( j \in I \) MaxVectors. The discussions in this paper can be extended to find

\[
\left\{ j \mid \sum_{k=1}^{m_j} p_k^{(j)} = \min_{1 \leq i \leq n} \left( \sum_{k=1}^{m_i} p_k^{(i)} \right) \right\}, \text{ or} \tag{2}
\]

\[
\left\{ j \mid \sum_{k=1}^{m_j} p_k^{(j)} = \max_{1 \leq i \leq n} \left( \sum_{k=1}^{m_i} p_k^{(i)} \right) \right\}. \tag{3}
\]
We call vectors \( p^{(j)} \) for \( j \) in the set of (2) **MinVectors**.

To find the set \( I \) in (1), one may compute \( \sum_{k=1}^{n} p_k^{(i)} = 1, \ldots, n \), and obtain the set \( I \) by \( n - 1 \) comparisons of the computed results. However, using a floating-point arithmetic has a pitfall. Notation \( u \) denotes the unit roundoff (e.g., \( u = 2^{-53} \)) for binary64 in IEEE 754. Let \( \text{fl}(\cdot) \) be the result of the floating-point operations. All operations inside the parentheses are executed by ordinary floating-point arithmetic rounding to the nearest (roundTiesToEven defined in IEEE 754). Rounding errors may occur in each arithmetic operation if we use numerical computations. We show the problems in the following three simple examples:

\[
\begin{align*}
    p^{(1)} &= (1 + 2u, 1 + 2u, 2u)^T, & p^{(2)} &= (2, 2 + 4u, -2 + 4u)^T, \\
    q^{(1)} &= (1, 2u, 0, \ldots, 0)^T \in \mathbb{F}^m, & q^{(2)} &= (1, u, u, \ldots, u)^T \in \mathbb{F}^m,
\end{align*}
\]

and

\[
\begin{align*}
    r^{(1)} &= (2, 10u, -3u, -2u)^T, & r^{(2)} &= (2, -2u, 2u, 3u)^T, & r^{(3)} &= (2, 6u, -2u, -2u)^T.
\end{align*}
\]

In these examples, we define

\[
\text{fl}\left(\sum_{i=1}^{m} x_i\right) := \text{fl}(\cdots((x_1 + x_2) + x_3) + \cdots) + x_m \quad \forall x \in \mathbb{F}^m.
\]

For (4) to (6), the results for the exact computations are

\[
\begin{align*}
    \sum_{i=1}^{3} p_i^{(1)} &= 2 + 6u, & \sum_{i=1}^{3} p_i^{(2)} &= 2 + 8u, \\
    \sum_{i=1}^{m} q_i^{(1)} &= 1 + 2u, & \sum_{i=1}^{m} q_i^{(2)} &= 1 + (m - 1)u, \\
    \sum_{i=1}^{4} r_i^{(1)} &= 2 + 5u, & \sum_{i=1}^{4} r_i^{(2)} &= 2 + 3u, & \sum_{i=1}^{4} r_i^{(3)} &= 2 + 2u,
\end{align*}
\]

while those for the numerical computations are

\[
\begin{align*}
    \text{fl}\left(\sum_{i=1}^{3} p_i^{(1)}\right) &= 2 + 8u, & \text{fl}\left(\sum_{i=1}^{3} p_i^{(2)}\right) &= 2 + 4u, \\
    \text{fl}\left(\sum_{i=1}^{m} q_i^{(1)}\right) &= 1 + 2u, & \text{fl}\left(\sum_{i=1}^{m} q_i^{(2)}\right) &= 1, \\
    \text{fl}\left(\sum_{i=1}^{4} r_i^{(1)}\right) &= 2, & \text{fl}\left(\sum_{i=1}^{4} r_i^{(2)}\right) &= 2 + 4u, & \text{fl}\left(\sum_{i=1}^{4} r_i^{(3)}\right) &= 2 + 8u.
\end{align*}
\]

Therefore, for (4), (5) for \( m \geq 4 \), and (6),

\[
\begin{align*}
    \sum_{i=1}^{3} p_i^{(1)} &< \sum_{i=1}^{3} p_i^{(2)}, & \text{fl}\left(\sum_{i=1}^{3} p_i^{(1)}\right) &> \text{fl}\left(\sum_{i=1}^{3} p_i^{(2)}\right),
\end{align*}
\]
Hence, we obtain the incorrect results using the numerical computations from the examples (4) to (6). Example (4) shows that even the computations for a vector of length three make a wrong decision. Example (5) for a large \( m \) implies that even if the gap between the sums is large, the judgment may be wrong. Example (6) indicates that the result may give the diametrically opposite judgment. We can apply accurate numerical algorithms for the summation [2, 3, 4, 5, 6, 7, 8, 9] to overcome the problem caused by rounding errors. For example, Algorithm 7.4 in [9] produces a nearest floating-point number to an exact sum. For example, real numbers \( 1 + 3u \) and \( 1 + 5u \) are rounded to \( 1 + 4u \in \mathbb{F} \) according to the rounding to nearest (ties to even rule). The problem remains unsolved using the accurate summation algorithm.

The other possibility is to accurately compare two sums. If the numerical computations produce the correct sign of \( \sum_{k=1}^{m_i} p_k^{(i)} - \sum_{k=1}^{m_j} p_k^{(j)} \), which one is bigger is exactly known, that is, \( \sum_{k=1}^{m_i} p_k^{(i)} \) or \( \sum_{k=1}^{m_j} p_k^{(j)} \). One fast and accurate algorithm produces the correct sign of the sum (Algorithm 4.1 in [9], with details introduced in Section 2.2). If we make the vector

\[
q := \left( p_1^{(i)}, p_2^{(i)}, \ldots, p_{m_i}^{(i)}, -p_1^{(j)}, -p_2^{(j)}, \ldots, -p_{m_j}^{(j)} \right)^T \in \mathbb{F}^{m_i + m_j},
\]

Algorithm 4.1 in [9] for \( q \) produces the correct sign of \( \sum_{k=1}^{m_i} p_k^{(i)} q_k \). Therefore, \( \sum_{k=1}^{m_i} p_k^{(i)} \) and \( \sum_{k=1}^{m_j} p_k^{(j)} \) can be exactly compared. Let \( p^{(1)} \) be a tentative \textbf{MaxVector}. We compare \( p^{(1)} \) and \( p^{(3)} \) using the abovementioned method and determine the next tentative \textbf{MaxVector}. The \textbf{MaxVectors} can finally be obtained by comparing \( n-1 \) times using the accurate summation algorithm. This approach is free from the problem of rounding errors. However, we need not always apply the accurate summation algorithm. If the sums are not close, the comparison of two vectors using numerical computations is correct in many cases.

We propose herein an adaptive numerical algorithm finding \textbf{MaxVectors}. The proposed algorithm works with as much cost as necessary to obtain the correct result. First, we apply a floating-point filter that gives sufficient conditions for correctness using only numerical computations. This technique is used for geometric predicates [10, 11, 12, 13, 14]. Next, we adopt Algorithm 4.1 in [9] to the search for \textbf{MaxVectors}.

For \( a, b \in \mathbb{F} \), in the absence of overflow and underflow, a product of two floating-point numbers \( a \cdot b \in \mathbb{R} \) can be transformed into \( c + d \) such that

\[
a \cdot b = c + d, \quad c = \text{fl}(a \cdot b), \quad d = a \cdot b - \text{fl}(a \cdot b), \quad c, d \in \mathbb{F}.
\]  

(7)

For example, we can obtain \( c \) and \( d \) using the error-free transformation by Dekker and Veltkamp [15] and Karp and Markstein [16]. For \( x, y \in \mathbb{F}^n \), the dot product \( x^T y \) is transformed into an unevaluated sum of \( 2n \) floating-point numbers. Therefore, we can find the maximum of the dot products from the given \( n \) pairs of vectors. Many problems can boil down to the sum of floating-point numbers applying the error-free transformation (7) (e.g.,
searching for the maximum of matrix determinants, maximum values of polynomials, far-
thest points from a point, most distant points from a plane, etc.). Therefore, the proposed
algorithm can be applied to many scientific problems.

The remainder of this paper is organized as follows: Section 2 introduces the notations
and previous works; Section 3 presents the proposed filter and robust method for searching
MaxVectors; Section 4 shows the validity of the proposed methods; Section 5 explains the
extension of the proposed method for the overflow case; and Section 6 presents numerical
examples to illustrate the efficiency of the proposed method.

2. Notation and Previous Work

2.1. Notation

Let $\nabla(\cdot)$ and $\Delta(\cdot)$ be the result of the floating-point operations. All operations inside the
parentheses are executed by ordinary floating-point arithmetic in rounding-downwards -
upwards (roundTowardNegative and roundTowardPositive, respectively, defined in IEEE
754). The underflow unit is denoted by $S_{\text{min}}$. The maximum floating-point number is repre-
sented by $F_{\text{max}}$. The minimum of the positive normalized floating-point number is denoted
by $F_{\text{min}}$ (e.g., $S_{\text{min}} = 2^{-1022}$, $F_{\text{max}} = 2^{1024} (1 - u)$, and $F_{\text{min}} = 2^{-1022}$ for binary64 in IEEE
754). Let $a \mathbb{Z}$ denote the set of the integral multiple of $a \in \mathbb{R}$ (i.e., $a \mathbb{Z} = \{az | z \in \mathbb{Z}\}$). Define
functions $\text{ufp}(a)$ and $\text{sign}(a)$ for $a \in \mathbb{R}$ as follows:

$$\text{ufp}(a) = \begin{cases} 0 & (\text{if } a = 0) \\ 2^{\lfloor \log_2 |a| \rfloor} & (\text{otherwise}) \end{cases}, \quad \text{sign}(a) = \begin{cases} 0 & (\text{if } a = 0) \\ 1 & (\text{if } a > 0) \\ -1 & (\text{otherwise}) \end{cases}$$

We write algorithms using a MATLAB-style notation throughout the paper.

2.2. Previous Work

First, we will introduce the lemmas used for proofs in this paper.

Lemma 1. For $a \in \mathbb{R}$ s.t. $|a| \leq F_{\text{max}}$ and $a \in S_{\text{min}} \mathbb{Z}$,

$$a \in \mathbb{F} \iff \exists k \in \mathbb{Z} \text{ s.t. } |a| \leq 2^k, \ a \in u 2^k \mathbb{Z}.$$  

This lemma was obtained from the definition of the binary floating-point numbers in
IEEE 754.

Lemma 2 ([2]). For $a, b \in \mathbb{F}$, in the absence of an overflow,

$$|\nabla(a + b) - (a + b)| \leq u \cdot \text{ufp}(a + b).$$  

Next, we will introduce the algorithm for the error-free splitting of a floating-point num-
ber into two floating-point numbers.
Algorithm 1 ([2, Algorithm 3.2]). Error-free splitting of a floating-point number into two floating-point numbers when no overflow occurs.

**Input:** \( a, \sigma \in \mathbb{F} \)

**Require:** \( \sigma \geq |a| \)

**Output:** \( x, y \in \mathbb{F} \)

1: function \([x, y] = \text{ExtractScalar}(\sigma, a)\)
2: \( x = \text{fl}((a + \sigma) - \sigma); \quad y = \text{fl}(a - x); \)
3: end function

If we apply Algorithm 1 to a floating-point number \( a \), then \( a \) is split into \( x \) and \( y \) (Fig. 1).

![Algorithm 1 Image](image)

Figure 1: Image of Algorithm 1 for \( \sigma = 2^k \)

We will now introduce the algorithm for the error-free transformation of a floating-point vector.

Algorithm 2 ([2, Algorithm 3.4]). Error-free transformation of \( p \in \mathbb{F}^n \) into \( \tau \in \mathbb{F} \) and \( p' \in \mathbb{F}^n \) such that \( \sum_{i=1}^n p_i = \tau + \sum_{i=1}^n p'_i \) when no overflow occurs.

**Input:** \( \sigma \in \mathbb{F}, \ p \in \mathbb{F}^n \)

**Require:** \( \sigma \geq 2^k \cdot \max_i |p_i| > n \cdot \max_i |p_i|, \ k \in \mathbb{Z} \)

**Output:** \( \tau \in \mathbb{F}, \ p' \in \mathbb{F}^n \)

1: function \([\tau, p'] = \text{ExtractVector}(\sigma, p)\)
2: for \( i = 1 : n \)
3: \( [q_i, p'_i] = \text{ExtractScalar}(\sigma, p_i); \quad \)▷ Algorithm 1
4: end for
5: \( \tau = \text{fl} \left( \sum_{i=1}^n q_i; \right) \);  
6: end function

Note that Line 5 in Algorithm 2 can be computed in any order without rounding errors. If we apply Algorithm 2 to a floating-point vector \( p \), then Algorithm 1 is applied to all \( p_i \), and \( \tau \) is the sum of all high-order parts (Fig. 2).

Next, we introduce the algorithm to compute \( 2^{\lfloor \log_2 |a| \rfloor} \) for an input floating-point number \( a \neq 0 \) when no overflow occurs. The following algorithm can produce \( 2^{\lfloor \log_2 |a| \rfloor} \) without taking the binary logarithm:

Algorithm 3 ([2, Algorithm 3.6]). Computation of \( 2^{\lfloor \log_2 |a| \rfloor} \) for \( a \neq 0 \) without computing the binary logarithm when no overflow occurs.
Input: $a \in \mathbb{F}$
Require: $a \neq 0$
Output: $L \in \mathbb{F}$

1: function $L = \text{NextPowTwo}(a)$
2: $b = \text{fl}(u^{-1}a)$; $L = \text{fl}((b + a) - b)$;
3: if $L == 0$, $L = |a|$; end if
4: end function

Finally, we introduce the algorithm for the rigorous computations of the sign of the sum of all elements of a floating-point vector when no overflow occurs.

Algorithm 4 ([9, Algorithm 4.1]). Rigorous computations of the sign of a sum when no overflow occurs.

Input: $p \in \mathbb{F}^n$
Require: $2^{\lceil \log_2(n+2) \rceil}u < 1$
Output: $s \in \mathbb{F}$

1: function $s = \text{AccSign}(p)$
2: $\mu = \max_{1 \leq i \leq n} |p_i|$;
3: if $\mu == 0$, $s = 0$; return; end if
4: $M = \text{NextPowTwo}(n + 2)$; $t = 0$; \hspace{1cm} ▶ Algorithm 3
5: $\sigma = \text{fl}(M \cdot u \cdot \sigma')$;
6: while $1$ do
7: $[\tau, \rho] = \text{ExtractVector}(\sigma, p)$; \hspace{1cm} ▶ Algorithm 2
8: $t = \text{fl}(t + \tau)$; $\sigma' = \sigma$;
9: $\sigma = \text{fl}(M \cdot u \cdot \sigma')$;
10: if $|t| \geq \sigma \& \& \sigma' \leq F_{\text{min}}$, break; end if
11: end while
12: $s = \text{sign}(t)$;
13: end function

Assume $q := (p_1^{(i)}, p_2^{(i)}, \ldots, p_m^{(i)}, -p_1^{(i)}, -p_2^{(i)}, \ldots, -p_m^{(i)})^T \in \mathbb{F}^{m_i+m_j}$ for $p^{(i)} \in \mathbb{F}^{m_i}$, $p^{(j)} \in \mathbb{F}^{m_j}$. Let $s$ be the result obtained by $\text{AccSign}(q)$ (Algorithm 4). Then,

$$s = \text{sign} \left( \sum_{k=1}^{m_i} p_k^{(i)} - \sum_{k=1}^{m_j} p_k^{(j)} \right).$$
Hence

\[ s \leq 0 \iff \sum_{k=1}^{m_i} p_k^{(i)} \leq \sum_{k=1}^{m_j} p_k^{(j)}. \]

Algorithm 4 is useful in obtaining the rigorous result of the comparison between the sum of the two vectors. Therefore, we can search for MaxVectors using the following algorithm based on the comparison using Algorithm 4:

**Algorithm 5.** Searching for MaxVectors based on comparison using Algorithm 4 when no overflow occurs.

**Input:** \( p^{(i)} \in \mathbb{R}^{m_i}, i = 1, \ldots, n \)

**Require:** \( 2^{\lceil \log_2 (2 \max m_i + 2) \rceil} u < 1 \)

**Output:** \( I \in \mathbb{N}^N \)

1: function \( I = \text{SearchMaxVecAS}(p^{(1)}, \ldots, p^{(n)}) \)
2: \( N = 1; i = 1; I_1 = 1; \) \( \triangleright I \in \mathbb{F}^n \)
3: for \( j = 2 : n \) do
4: \( s = \text{AccSign}([p^{(i)}; -p^{(j)}]); \) \( \triangleright [p^{(i)}; -p^{(j)}] \in \mathbb{R}^{m_i + m_j}. \) Algorithm 4
5: if \( s < 0 \) then
6: \( N = 1; I_1 = j; i = j; \)
7: else if \( s == 0 \) then
8: \( N = N + 1; I_N = j; \)
9: end if
10: end for
11: \( I = I(1 : N); \)
12: end function

MATLAB CellArray \( p \) s.t., \( p[i] = p^{(i)} \) can be used for the arguments of Algorithm 5. We designed Algorithm 5 to compare two vectors in a recursive order. Let \( I \) be the result obtained by Algorithm 5. \( p^{(i)} \) for all \( i \in I \) are MaxVectors.

The computational time for Algorithm 5 depends on the order of vectors. To see this, let three vectors be

\[
\begin{align*}
p^{(1)} &:= (u^{-1}, u^{-1}, u^{-1}, u^{-1})^T \in \mathbb{F}^4, \\
p^{(2)} &:= (2u, u^3 - u^{-2}, u^2, -u^{-1})^T \in \mathbb{F}^4, \\
p^{(3)} &:= (u^{-2} + 2u^{-1}, u, u, -u^{-2} - 2u^{-1})^T \in \mathbb{F}^4.
\end{align*}
\]

Because

\[ 4u^{-1} = \sum_{i=1}^{4} p_i^{(1)} \Rightarrow \sum_{i=1}^{4} p_i^{(2)} = \sum_{i=1}^{4} p_i^{(3)} = 2u, \]

the computational time for \text{AccSign}([p^{(1)}; p^{(2)}]) is shorter than that for \text{AccSign}([p^{(2)}; p^{(3)}]). Hence, the computational time for \text{SearchMaxVecAS}(p^{(1)}, p^{(2)}, p^{(3)}) is faster than that for \text{SearchMaxVecAS}(p^{(3)}, p^{(2)}, p^{(1)}). Thus, Algorithm 5 can become inefficient depending on the vector order. Therefore, we develop an accurate algorithm, whose cost is independent of the vector order.
3. Proposed Method

We designed the method to repeatedly apply algorithms to the vectors and exclude the vectors that are not \textbf{MaxVectors} from the computations. We present two algorithms herein: one is a simple filter, and the other is a robust algorithm used to search for \textbf{MaxVectors} employs Algorithm 2. Let the initial set be $\tilde{I} := \{1, 2, \ldots, n\}$.

3.1. Filtering without High-accuracy Computations

We now search for vectors that are not \textbf{MaxVectors} without robust computations and exclude them. We made a filter using directed rounding. It provided a sufficient condition for comparison correctness.

Define $D_i, U_i \in F$ for all $i$ and $D_{\text{max}} \in F$ as

\[
D_i := \text{fl}_\triangleleft \left( \sum_{k=1}^{m} p_k^{(i)} \right), \quad U_i := \text{fl}_\triangleright \left( \sum_{k=1}^{m} p_k^{(i)} \right), \quad D_{\text{max}} := \max_{1 \leq i \leq n} D_i.
\]

Note that the computational order in $\text{fl}_\triangleleft (\cdot)$ and $\text{fl}_\triangleright (\cdot)$ has no restrictions. We accept recursive, blockwise, and pairwise orders. It holds that

\[
D_i \leq \sum_{k=1}^{m} p_k^{(i)} \leq U_i \quad \forall i.
\]

Figure 3 shows an image of $D_{\text{max}}$ and intervals $[D_i, U_i]$.

Vectors $\{p^{(i)} \mid U_i < D_{\text{max}}\}$ cannot be \textbf{MaxVectors}; hence, we can exclude $\{i \mid U_i < D_{\text{max}}\}$ from $\tilde{I} = \{1, 2, \ldots, n\}$. After this process is executed for $i = 1, \ldots, n$, if

\begin{itemize}
  \item $|\bar{I}| = |\{i \mid U_i \geq D_{\text{max}}\}| = 1$, or
  \item $D_i = U_i$ for all $i \in \bar{I} = \{i \mid U_i \geq D_{\text{max}}\}$,
\end{itemize}

then $p^{(i)}$ for all $i \in \bar{I}$ are \textbf{MaxVectors}. A similar discussion can be done for \textbf{MinVectors}.

Note that it is not always necessary to compute both the upper and lower bounds of all the vectors’ sum. For example, computing $D_i$ is not necessary if $\exists j \text{ s.t. } U_j < D_{\text{max}}$.

Finally, we present an algorithm for a filter to exclude vectors that are not \textbf{MaxVectors} without robust computations.
Algorithm 6. Filtering for excluding vectors that are not MaxVectors without robust computations.

Input: $p^{(i)} \in \mathbb{R}^m$, $i = 1, \ldots, n$

Output: $\bar{I} \in \mathbb{R}^N$

1: function $[\bar{I}, \text{flag}] = \text{Filtering}(p^{(1)}, \ldots, p^{(n)})$
2: \hspace{1cm} $D_1 = \text{fl}_\nu (\sum_{k=1}^m p_k^{(1)})$; $U_1 = \text{fl}_\kappa (\sum_{k=1}^m p_k^{(1)})$; $\triangleright$ $D, U \in \mathbb{R}^n$
3: \hspace{1cm} $D_{\max} = D_1$;
4: \hspace{1cm} for $i = 2 : n$ do
5: \hspace{2cm} $D_i = \text{fl}_\nu (\sum_{k=1}^m p_k^{(i)})$; $U_i = \text{fl}_\kappa (\sum_{k=1}^m p_k^{(i)})$;
6: \hspace{2cm} if $D_i > D_{\max}$, $D_{\max} = D_i$; end if $\triangleright$ Update $D_{\max}$.
7: end for
8: \hspace{1cm} if $U_i \geq D_{\max}$, $N = N + 1$; $\bar{I}_N = i$; end if $\triangleright$ $\bar{I} \in \mathbb{R}^N$
9: end for
10: $\bar{I} = \bar{I}(1 : N)$; $\text{flag} = 1$;
11: if $N == 1$, return; end if $\triangleright$ Searching for MaxVectors is completed.
12: for $i = 1 : N$ do
13: \hspace{1cm} if $D(\bar{I}_i) \neq U(\bar{I}_i)$, $\text{flag} = 0$; break; end if
14: end for
17: end function

Let $\bar{I}$ and $\text{flag}$ be the results obtained by Algorithm 6. Then, $p^{(i)}$ for all $i \in \{1, 2, \ldots, n\} \setminus \bar{I}$ are not MaxVectors. If $\text{flag} = 1$, then $p^{(i)}$, $i \in \bar{I}$ are MaxVectors; otherwise, we must apply the robust algorithm introduced in subsection 3.2.

An overflow never causes a problem in this filter. The standard property of the floating-point arithmetic with directed rounding yields

\[ D_i = \text{fl}_\nu \left( \sum_{k=1}^n p_k^{(i)} \right) \neq \text{Inf}, \quad U_i = \text{fl}_\kappa \left( \sum_{k=1}^n p_k^{(i)} \right) \neq -\text{Inf} \quad \forall p^{(i)} \in \mathbb{R}^n. \]

Hence, we consider the case of

\[ D_i = \text{fl}_\nu \left( \sum_{k=1}^n p_k^{(i)} \right) = -\text{Inf}, \quad U_i = \text{fl}_\kappa \left( \sum_{k=1}^n p_k^{(i)} \right) = \text{Inf}. \]

If $D_i = -\text{Inf}$ for some $i$, but no overflow occurs for the other $i$, then, $D_{\max} \neq -\text{Inf}$ (Line 6, Algorithm 6). For the following cases:

\[ D_i = -\text{Inf} \quad \forall i \quad (D_{\max} = -\text{Inf}) \quad \text{or} \quad U_i = \text{Inf} \quad \exists i, \]

$U_i \geq D_{\max}$ in Line 10 of Algorithm 6 is true; thus, the vectors are not excluded by the filter.

### 3.2. Filtering with Robust Computations

We now consider accurately searching for MaxVectors by repeatedly applying Algorithm 2. Here, let $\bar{I}$ and $\text{flag}$ be the results obtained by Algorithm 6 and $\text{flag} = 0$, respectively.
Let $p^{(v,i)}$ denote $p^{(i)}$ in the $v$-th iteration and $\bar{I}^{(v)}$, $\mu^{(v)}$, $\sigma^{(v)}$, $t_i^{(v)}$, $\tau_i^{(v)}$, $T_i^{(v)}$ denote a variable in the $v$-th iteration. Define

$$\bar{I}^{(0)} := I, \quad M := \max_{i \in I} \left( 0.75 \cdot 2^{\lceil \log_2 (m+1) \rceil} \right), \quad m := \max_{i \in I} m_i$$

and $p^{(0,i)} := p^{(i)} \in \mathbb{F}^m$, for all $i \in \bar{I}^{(0)}$. Assume

$$2 \cdot 2^{\lceil \log_2 (m+1) \rceil} \cdot 2^{\lceil \log_2 m \rceil} \cdot \mu \leq 1.$$  \hspace{1cm} (9)

For $v = 1$, we determine $\mu^{(1)} \in \mathbb{F}$ as

$$\mu^{(1)} := \max_{i \in \bar{I}^{(0)}} \max_{1 \leq j \leq m_i} \left| p_j^{(0,i)} \right| = \max_{i \in I} \left| p_j^{(0)} \right|$$

and $\sigma^{(1)} \in \mathbb{F}$ using $M$ in (8) as

$$\sigma^{(1)} := \max \left( M \cdot 2^{\lceil \log_2 \mu^{(1)} \rceil} \right).$$  \hspace{1cm} (10)

Note that $\mu^{(1)} \neq 0$ because flag = 0. Here, we execute

$$\left[ t_i^{(1)}, p^{(1,i)} \right] = \text{ExtractVector}(\sigma^{(1)}, p^{(0,i)}) \quad \forall i \in \bar{I}^{(0)},$$

and define

$$\tau_i^{(1)} := t_i^{(1)} \quad \forall i \in \bar{I}^{(0)}.$$  \hspace{1cm} (11)

Figure 4 shows an image of $\tau_i^{(1)}$, $p^{(1,i)}$ for all $i \in \bar{I}^{(0)}$.

If

$$T_i^{(1)} := \max_{i \in \bar{I}^{(0)}} \left( \tau_i^{(1)} - \max_{i \in I} \tau_i^{(1)} \right) < \max_{i \in \bar{I}^{(0)}} \left( -2m \mu \cdot \text{ufp} \left( \sigma^{(1)} \right) \right) = \max_{i \in \bar{I}^{(0)}} \left( -4m \mu \left( \frac{\sigma^{(1)}}{3} \right) \right)$$

is satisfied, $p^{(i)}$ for such $i$ are not MaxVectors. This is proven in Theorem 8 of Section 4. Hence, we exclude $i$ from $I^{(0)}$ and update to $I^{(1)}$ as

$$I^{(1)} := I^{(0)} \backslash \left\{ i \in I^{(0)} \mid T_i^{(1)} < \max_{i \in \bar{I}^{(0)}} \left( -4m \mu \left( \frac{\sigma^{(1)}}{3} \right) \right) \right\}. $$  \hspace{1cm} (12)

Note that no rounding error exists in the right and left sides for $i \in I^{(1)}$ of (13).

For $v \geq 2$, if $| \bar{I}^{(v-1)} | = 1$, then $p^{(i)}$, $i \in \bar{I}^{(v-1)}$ is a MaxVector. If $| \bar{I}^{(v-1)} | \neq 1$, let $\mu^{(v)}$ be

$$\mu^{(v)} := \begin{cases} \max_{i \in \bar{I}^{(v-1)}} \left| t_i^{(v-1)} \right| & (\text{if } \exists T_i^{(v-1)} \neq 0) \\ \max_{i \in \bar{I}^{(v-1)}} \left| p_j^{(v-1,i)} \right| & (\text{otherwise}) \end{cases}\quad (13)$$

If $\mu^{(v)} = 0$, then $p^{(i)}$, $\forall i \in \bar{I}^{(v-1)}$ are MaxVectors. If $\mu^{(v)} \neq 0$, then we determine

$$\sigma^{(v)} := \max \left( M \cdot 2^{\lceil \log_2 \mu^{(v)} \rceil} \right).$$  \hspace{1cm} (14)
We execute
\[
[t^{(ν)}_i, p^{(ν, i)}] = \text{ExtractVector}(σ^{(ν)}, p^{(ν-1, i)}) \quad \forall i \in \bar{I}^{(ν-1)}
\]  
and update \(τ_i\) for all \(i \in \bar{I}^{(ν-1)}\) as
\[
τ^{(ν)}_i := \text{fl}(T^{(ν-1)}_i + t^{(ν)}_i).
\]  
We have proven that no rounding error exists in (18) in Theorem 7 of Section 4. We can then search for vectors that are not \textbf{MaxVectors} by the same condition
\[
T^{(ν)}_i := \text{fl}\left(τ^{(ν)}_i - \max_{i \in \bar{I}^{(ν-1)}} τ^{(ν)}_i\right) < \text{fl}\left(-2mu \cdot \text{ufp}(σ^{(ν)})\right) = \text{fl}\left(-4mu \left(\frac{σ^{(ν)}}{3}\right)\right)
\]  
as (13). Hence, \(\bar{I}\) is updated by
\[
\bar{I}^{(ν)} := \bar{I}^{(ν-1)} \setminus \left\{ i \in \bar{I}^{(ν-1)} \mid T^{(ν)}_i < \text{fl}\left(-4mu \left(\frac{σ^{(ν)}}{3}\right)\right) \right\}.
\]  
We found no rounding error in the right and left sides for \(i \in \bar{I}^{(ν)}\) of (19). The latter is proven in Theorem 6 of Section 4. We iterate the same processes from (15) to (20) until \(\bar{I}^{(ν)} = 1\) or \(μ^{(ν+1)} = 0\). Finally, we present an algorithm to accurately search for \textbf{MaxVectors}.

\textbf{Algorithm 7.} Searching for \textbf{MaxVectors} correctly when no overflow occurs.

**Input:** \(p^{(i)} \in \mathbb{Z}_m, i = 1, \ldots, n, \bar{I} \in \mathbb{Z}_L\)

**Require:** \(\bar{I} = \text{Filtering}(p^{(1)}, \ldots, p^{(n)}), 2 \cdot 2^{\left[\log_2(m+1)\right]} \cdot 2^{\left[\log_2(m)\right]} \cdot u \leq 1\)

**Output:** \(\bar{I} \in \mathbb{Z}_N\)
3.3. Verified Numerical Computations for Searching for Vectors with the Maximum Sum

The following algorithm is obtained from the abovementioned discussions:

**Algorithm 8.** Verified numerical computations for the search for \textit{MaxVectors} when no overflow occurs.

\textbf{Input:} $p^{(i)} \in \mathbb{R}^{m_i}, i = 1, \ldots, n$

\textbf{Require:} $2 \cdot 2^{[\log_2(\max m_i + 1)]}, 2^{[\log_2\max m_i]}u \leq 1$

\textbf{Output:} $I \in \mathbb{R}^N$

1: \textbf{function} $\bar{I} = \text{SearchMaxVec}(p^{(1)}, \ldots, p^{(n)}, \bar{I})$
2: \quad $\epsilon = \text{fl}(\bar{4}mu)$; $M = \text{fl}(0.75 \cdot \text{NextPowTwo}(m + 1))$; $N = L$; \quad $\triangleright$ Algorithm 3
3: \quad $\mu = \max_{i \leq I, 1 \leq j \leq m_i} |p^{(i)}_j|$; $\sigma = \text{fl}(M \cdot \text{NextPowTwo}(\mu))$; \quad $\triangleright$ $\mu \neq 0$
4: \quad \textbf{for} $i = 1 : N$, $[\tau_i, p^{(i)}_j] = \text{ExtractVector}(\sigma, p^{(i)}_j)$; \textbf{end for} \quad $\triangleright \tau \in \mathbb{R}^k$. Algorithm 2
5: \quad $\tau_{\max} = \max_{1 \leq i \leq N} \tau_i$;
6: \quad \textbf{while} 1 do
7: \quad \quad $\delta = \text{fl}(\epsilon \cdot (\sigma / 3))$; $j = 0$; \quad $\triangleright \delta = -\bar{4}mu\sigma / 3$.
8: \quad \quad \textbf{for} $i = 1 : N$ do
9: \quad \quad \quad $\tau_{\max} = \max_{1 \leq j \leq m_i} |p^{(i)}_{\max}|$
10: \quad \quad \quad if $\tau_i \geq \delta$, $j = j + 1$; \quad $\tau_j = \tau_i$; $\bar{I}_j = I_i$; \textbf{end if}
11: \quad \quad \textbf{end for}
12: \quad \quad $N = j$;
13: \quad \quad if $N == 1$, \textbf{break}; \textbf{end if} \quad $\triangleright$ Searching for \textit{MaxVectors} is completed.
14: \quad \quad $\mu = \max_{1 \leq i \leq N} |\tau_i|$;
15: \quad \quad \textbf{if} $\mu == 0$ \textbf{then}
16: \quad \quad \quad $\mu = \max_{i \in [1 : N]} |p^{(i)}_j|$ \quad $\triangleright$ Searching for \textit{MaxVectors} is completed.
17: \quad \quad \quad \textbf{if} $\mu = 0$, \textbf{break}; \textbf{end if}
18: \quad \quad \textbf{end if}
19: \quad \quad $\sigma = \text{fl}(M \cdot \text{NextPowTwo}(\mu))$;
20: \quad \quad \textbf{for} $i = 1 : N$ do
21: \quad \quad \quad $[\tau, p^{(i)}_j] = \text{ExtractVector}(\sigma, p^{(i)}_j)$;
22: \quad \quad \quad $\tau_{\max} = \max_{1 \leq j \leq m_i} \tau_i$;
23: \quad \quad \textbf{end for}
24: \quad \textbf{end while}
25: \quad $\bar{I} = \bar{I}(1 : N)$;
26: \textbf{end function}
Let $I$ be the results obtained by Algorithm 8; $\mathbf{p}^{(i)}$ for all $i \in I$ are \textbf{MaxVectors}. Moreover, we obtain the set in (2) if we execute Algorithm 8 to $-\mathbf{p}^{(i)} \in \mathbb{F}^m$, $i = 1, \ldots, n$. The alternative is to replace $I$ and $\tau_i$ by

$$F^{(v)} := F^{(v-1)} \setminus \left\{ i \in F^{(v-1)} \mid \tau_i^{(v-1)} > \text{fl} \left( 4\text{u} \left( \frac{\sigma^{(v)}}{3} \right) \right) \right\}, \quad T_i^{(v)} := \text{fl} \left( \tau_i^{(v)} - \min_{i \in F^{(v-1)}} \tau_i^{(v)} \right)$$

for $v = 1, 2, \ldots$ instead of (19) and (20). In this case, the following changes are implemented in Algorithm 7.

5: $\tau_{\max} = \max_{1 \leq i \leq n} \tau_i \quad \Rightarrow \quad \tau_{\min} = \min_{1 \leq i \leq N} \tau_i$
9: $\tau_i = \text{fl}(\tau_i - \tau_{\max}) \quad \Rightarrow \quad \tau_i = \text{fl}(\tau_i - \tau_{\min})$
10: $\tau \geq \delta \quad \Rightarrow \quad \tau \leq \delta$
24: $\tau_{\max} = \max_{1 \leq i \leq n} \tau_i \quad \Rightarrow \quad \tau_{\min} = \min_{1 \leq i \leq N} \tau_i$

We obtain the set in (3) if we compare $|\sum_{k=1}^{m_i} p_k^{(i)}|$ for $i$ in the set in (1) and $|\sum_{k=1}^{m_j} p_k^{(j)}|$ for $j$ in the set in (2). By executing Algorithm 4 for

$$q := (p_1^{(i)}, \ldots, p_{m_i}^{(i)}, p_1^{(j)}, \ldots, p_{m_j}^{(j)}) \in \mathbb{R}^{m_i + m_j},$$

(21)

where $\mathbf{p}^{(i)}$ is a \textbf{MaxVector}, and $\mathbf{p}^{(j)}$ is a \textbf{MinVector}. We can perform the following comparison:

$$s \leq 0 \iff |\sum_{k=1}^{m_i} p_k^{(i)}| \leq |\sum_{k=1}^{m_j} p_k^{(j)}|, \quad s := \text{AccSign}(q).$$

Many common computations can be performed for \textbf{MaxVectors} and \textbf{MinVectors}; therefore, the additional cost for obtaining set (3) is not expensive. This is shown in the numerical examples in Section 6.

4. Validation of the Proposed Method

First, we introduce herein the basic properties of Algorithm 2.

\textbf{Theorem 3 ([17])}. Let $a \in \mathbb{F}$ and $M = 2^k \in \mathbb{F}$, $k \in \mathbb{N}$. Assume $\sigma = 1.5M \cdot 2^{[\log_2 |x|]} \in \mathbb{F}$.

Let $x, y \in \mathbb{F}$ be the results obtained by Algorithm 1. Then, in the absence of an overflow, the following properties are satisfied:

$$a = x + y, \quad x \in 2\text{u} \cdot \text{ufp}(|\sigma|), \quad |x| \leq (1.5M)^{-1} \sigma = 2^{[\log_2 |x|]}, \quad |y| \leq u \cdot \text{ufp}(\sigma).$$

\textbf{Theorem 4}. Let $p \in \mathbb{F}^n$ and $M = 2^k \in \mathbb{F}$, $k \in \mathbb{N}$. Assume that $M > 2^{[\log_2 n]}$ and $\sigma = 1.5M \cdot 2^{[\log_2 \max|p_i|]} \in \mathbb{F}$. Let $\tau \in \mathbb{F}$ and $p' \in \mathbb{F}^n$ be the results obtained by Algorithm 2. Then, in the absence of an overflow, the following properties are satisfied:

$$\sum_{i=1}^{n} p_i = \tau + \sum_{i=1}^{n} p_i', \quad \left| \sum_{i=1}^{n} p_i' \right| \leq nu \cdot \text{ufp}(\sigma), \quad \tau \in 2u \cdot \text{ufp}(\sigma \mathbb{Z}),$$

$$|\tau| \leq n(1.5M)^{-1} \sigma = n2^{[\log_2 \max|p_i|]} < \text{ufp}(\sigma).$$
The validity of Algorithm 8 is demonstrated as follows:

- $\sigma^{(v)}$ is monotonously decreased (Theorem 5), implying that \( \mu^{(v)} \) becomes zero in finite steps because the input arguments are floating-point vectors.
- No rounding error exists for non \( \text{MaxVectors} \) in Line 9 of Algorithm 7 (Theorem 6).
- No rounding error exists in Line 22 of Algorithm 7 (Theorem 7).
- The vectors satisfying conditions (13) and (19) are not \( \text{MaxVectors} \) (Theorem 8).

We also give theorems for these arguments.

**Theorem 5.** Assume (9). For $\sigma^{(v)}$ defined in (16) for $v \geq 1$, we obtain the following in the absence of an overflow

$$\sigma^{(v)} < \sigma^{(v-1)}$$

for $v \geq 2$.

**Proof.** Since (13), (14), (19), and (20), we obtain $-2mu \cdot \text{ufp}(\sigma^{(v-1)}) \leq T^{(v-1)} \leq 0$ for all $i \in \tilde{I}^{(v-1)}$. From this and (15), it holds that $\mu^{(v)} \leq 2mu \cdot \text{ufp}(\sigma^{(v-1)})$ for all $v \geq 2$. Thus, we have

$$\begin{align*}
\sigma^{(v)} &= M \cdot 2^{[\log_2 \mu^{(v)}]} \\
&\leq M \cdot 2^{[\log_2 2mu \cdot \text{ufp}(\sigma^{(v-1)})]} \\
&= M \cdot 2^{[\log_2 m]} \cdot 2^{4.5} = 2^{[\log_2 m]} \cdot 2^{4.5} \cdot 2^{\sigma^{(v-1)}} < \sigma^{(v-1)}
\end{align*}$$

for all $v \geq 2$ from (8) and (9). \( \square \)

**Theorem 6.** Assume (9). Then, for all $i \in \tilde{I}^{(v)}$, $v \geq 1$, we obtain the following in the absence of an overflow:

$$\begin{align*}
\text{fl} \left( \tau^{(v)}_i - \max_{j \in \tilde{I}^{(v-1)}} \tau^{(v)}_j \right) &\geq \text{fl} \left( -4mu \cdot \frac{\sigma^{(v)}}{3} \right) \Rightarrow \tau^{(v)}_i - \max_{j \in \tilde{I}^{(v-1)}} \tau^{(v)}_j \in \mathbb{F}.
\end{align*}$$

**Proof.** From (11), (17), and Theorem 4, for all $v \geq 1$, it holds that

$$\tau^{(v)}_i \in 2u \cdot \text{ufp}(\sigma^{(v)}) \mathbb{Z} \quad \forall i \in \tilde{I}^{(v-1)}. \quad (22)$$

Therefore, (12), (18), and Theorem 4 yield that for all $v \geq 1$, we obtain

$$\tau^{(v)}_i \in 2u \cdot \text{ufp}(\sigma^{(v)}) \mathbb{Z} \quad \forall i \in \tilde{I}^{(v-1)}. \quad (23)$$

This theorem is proven by contradiction; hence, hereafter, we assume that

$$\text{fl} \left( \tau^{(v)}_i - \max_{j \in \tilde{I}^{(v-1)}} \tau^{(v)}_j \right) \notin \mathbb{F} \quad (24)$$

for all $\tilde{I}^{(v-1)}$, $v \geq 1$. From (23), we have

$$\tau^{(v)}_i - \max_{j \in \tilde{I}^{(v-1)}} \tau^{(v)}_j \in 2u \cdot \text{ufp}(\sigma^{(v)}) \mathbb{Z} \quad \forall i \in \tilde{I}^{(v-1)}.$$
Therefore, due to (24) and the contrapositive of Lemma 2, we obtain
\[ \left| \tau_i^{(\nu)} - \max_{i \in \tilde{I}^{(\nu-1)}} \tau_i^{(\nu)} \right| > 2u fp(\sigma^{(\nu)}) , \]
which yields
\[ \lfloor \tau_i^{(\nu)} - \max_{i \in \tilde{I}^{(\nu-1)}} \tau_i^{(\nu)} \rfloor \geq 2u fp(\sigma^{(\nu)}) . \]
From this and (9), we have
\[ \lfloor \tau_i^{(\nu)} - \max_{i \in \tilde{I}^{(\nu-1)}} \tau_i^{(\nu)} \rfloor \geq -2u fp(\sigma^{(\nu)}) < -2m u \cdot uf p(\sigma^{(\nu)}) . \]
Assumption (24) must be false. Remark that
\[ -2m u \cdot uf p(\sigma^{(\nu)}) = -4m u \cdot \frac{\sigma^{(\nu)}}{3} = \lfloor -4m u \cdot \frac{\sigma^{(\nu)}}{3} \rfloor . \]
Therefore, the proof is completed. □

**Theorem 7.** Assume (9) and \( \mu^{(\nu)} \neq 0 \). Then, for all \( \nu \geq 2 \), \( i \in \tilde{I}^{(\nu-1)} \), no rounding error exists in the absence of an overflow:
\[ \tau_i^{(\nu)} := \lfloor T_i^{(\nu-1)} + t_i^{(\nu)} \rfloor . \]

**Proof.** We then have
\[ |T_i^{(\nu-1)}| \leq \mu^{(\nu)} \leq M^{-1} \sigma^{(\nu)} \quad \forall i \in \tilde{I}^{(\nu-1)} \] (25)
from (15), (16), and
\[ T_i^{(\nu)} \in 2u \cdot uf p(\sigma^{(\nu)}) \mathbb{Z} \quad \forall i \in \tilde{I}^{(\nu-1)} , \] (26)
because of (12), (13), (18), (19), and Theorem 4. Application of (25), Theorem 4, (8), (10), and (16) yields
\[ |T_i^{(\nu-1)} + t_i^{(\nu)}| \leq |T_i^{(\nu-1)} + t_i^{(\nu)}| \leq (m + 1)M^{-1} \sigma^{(\nu)} \]
\[ = 2(m + 1)2^{-\left\lceil \log_2(m+1) \right\rceil} \cdot uf p(\sigma^{(\nu)}) \leq 2u fp(\sigma^{(\nu)}) \quad \forall i \in \tilde{I}^{(\nu-1)} . \] (27)
Moreover, we obtain the following from (22) and (26):
\[ T_i^{(v-1)} + t_i^{(\nu)} \in 2u \cdot uf p(\sigma^{(\nu)}) \mathbb{Z} \quad \forall i \in \tilde{I}^{(\nu-1)} . \] (28)
Therefore, from (27) and (28), Lemma 1 gives
\[ \tau_i^{(\nu)} = \lfloor T_i^{(\nu-1)} + t_i^{(\nu)} \rfloor = T_i^{(\nu-1)} + t_i^{(\nu)} \quad \forall i \in \tilde{I}^{(\nu-1)} . \]
\[ \square \]

**Theorem 8.** Assume (9). Then, in the absence of an overflow, the conditions that are not \textit{MaxVectors} are given by (13) and (19).
Proof. From (11), (17), and Theorem 4, for \( n \geq 1 \), we have
\[
\sum_{k=1}^{m} p_k^{(v-1)} = t_i^{(v)} + \sum_{k=1}^{m} p_k^{(v)} \quad \text{and} \quad \sum_{k=1}^{m} |p_k^{(v)}| \leq m_2 u \cdot \text{ufp}(\sigma^{(v)}) \quad \forall i \in \bar{F}^{(v-1)}. \tag{29}
\]
Let \( \tau_i^{(v)} := \max_{i \in \bar{F}^{(v-1)}} \tau_i^{(v)} \), then using Theorem 6, 7, and (29) gives
\[
\sum_{k=1}^{m} p_k^{(i)} - \sum_{k=1}^{m} p_k^{(j)} = \left( t_i^{(1)} + \sum_{k=1}^{m} p_k^{(1)} \right) - \left( t_j^{(1)} + \sum_{k=1}^{m} p_k^{(1)} \right)
= \left( t_i^{(2)} + \sum_{k=1}^{m} p_k^{(2)} \right) - \left( t_j^{(2)} + \sum_{k=1}^{m} p_k^{(2)} \right) = \ldots
= \left( t_i^{(v)} + \sum_{k=1}^{m} p_k^{(v)} \right) - \left( t_j^{(v)} + \sum_{k=1}^{m} p_k^{(v)} \right)
\leq \tau_i^{(v)} + \sum_{k=1}^{m} p_k^{(v)} - \tau_j^{(v)} + \sum_{k=1}^{m} p_k^{(v)} \leq \tau_i^{(v)} - \tau_j^{(v)} + 2m_2 u \cdot \text{ufp}(\sigma^{(v)})
\]
for \( i \in \bar{F}^{(v-1)} \). Thus, if \( \tau_i^{(v)} - \tau_j^{(v)} < -2m_2 u \cdot \text{ufp}(\sigma^{(v)}) \), \( \sum_{k=1}^{m} p_k^{(i)} < \sum_{k=1}^{m} p_k^{(j)} \) is satisfied. \( \square \)

5. Overflow Case

In this section, we consider the overflow case in Algorithm 7. Let
\[
B := 2u^{-1}. \tag{30}
\]
If \( \sigma \) at Line 3 of Algorithm 7 is \( \text{Inf} \), we apply scaling by multiplying \( B^{-1} \). To check the rounding errors in \( \text{fl}(B^{-1} p_i^{(i)}) \), we define
\[
U^{(i)} := \{ k \mid p^{(i)} = \text{fl}(B \cdot (B^{-1} p_i^{(i)})) \} \quad \forall i \in \bar{F}^{(0)}.
\]
This implies that \( \text{fl}(B^{-1} p_i^{(i)}) \neq B^{-1} p_i^{(i)} \) for \( k \notin U^{(i)} \). We remove \( p_i^{(i)}, k \notin U^{(i)} \), and let vector \( p^{(i)} \in \mathbb{R}^{U^{(i)}} \). We set
\[
q^{(0, i)} := \text{fl}(B^{-1} p^{(i)}). \tag{31}
\]
We now drive the upper bound of \( p_i^{(i)}, k \notin U^{(i)} \). For a normalized floating-point number \( a \in \mathbb{F} \), the unit in the last place is \( 2u \cdot \text{ufp}(a) \). Therefore, \( p_i^{(i)}, k \notin U^{(i)} \) satisfies
\[
B^{-1} \cdot 2u \cdot \text{ufp}(p_i^{(i)}) \leq \frac{S_{\text{min}}}{2},
\]

and yields

$$ufp\left( P_{k}^{(i)} \right) \leq \frac{u_{k}^{-1} B \cdot S_{\text{min}}}{2}.$$  

A basic property of $ufp(a)$ for $0 \neq a \in \mathbb{R}$ gives

$$ufp(a) \leq |a| < 2ufp(a).$$

We then have

$$p_{k}^{(i)} \leq u_{k}^{-1} B \cdot S_{\text{min}} = u_{k}^{-2} \cdot S_{\text{min}}.$$  

(32)

For $\nu = 1$, we determine $\mu^{(1)} \in \mathbb{F}$ and $\sigma^{(1)} \in \mathbb{F}$ from $q^{(0,i)}$ as

$$\mu^{(1)} := \max_{i \in \bar{F}^{(0)}} \left| q_{k}^{(0,i)} \right|, \quad \sigma^{(1)} := \fl \left( M \cdot 2^{\lceil \log_{2} \mu^{(1)} \rceil} \right).$$  

(33)

Subsequently, from (30) and (31), we have $|u_{k}^{-1} a| \leq F_{\text{max}}/2$ in Algorithm 3. Thus,

$$|b| = \left| \fl \left( u_{k}^{-1} a \right) \right| \leq \frac{F_{\text{max}}}{2}, \quad \text{and} \quad L = \fl((b + a) - b) < F_{\text{max}}$$

hold such that no overflow occurs in Algorithm 3. In addition, (33), (8), and (9) give

$$\sigma^{(1)} = M \cdot 2^{\lceil \log_{2} \mu^{(1)} \rceil} \leq MB^{-1} \cdot 2^{\lceil \log_{2} \max \left| q_{k}^{(0,i)} \right| \rceil} \leq 2MB^{-1} \cdot \max \left| p_{k}^{(i)} \right| < F_{\text{max}},$$

and no overflow occurs in Algorithm 7.

We use $q$ instead of $p$ in Algorithm 7. From (32), if

$$\sigma^{(\nu)} \geq u_{k}^{-3} \cdot S_{\text{min}},$$

is satisfied for $\sigma$ at Line 19, then $\fl\left( p_{k}^{(i)} + \sigma^{(\nu)} \right) = \sigma^{(\nu)}$, $k \notin U^{(i)}$. If

$$\sigma^{(\nu)} < u_{k}^{-3} \cdot S_{\text{min}},$$

is satisfied for $\sigma$ at Line 19. There is a possibility that $\fl\left( p_{k}^{(i)} + \sigma^{(\nu)} \right) \neq \sigma^{(\nu)}$, $k \notin U^{(i)}$. In this case, we redefine $p_{k}^{(\nu,i)} \in \mathbb{F}^{m_{i}}$ and $\tau_{i}$ for all $i \in \bar{F}^{(\nu-1)}$ as

$$p_{k}^{(\nu,i)} := \begin{cases} p_{k}^{(i)} & \text{if } k \notin U^{(i)} \\ \fl\left( B_{k}^{(\nu,i)} \right) & \text{(otherwise)} \end{cases}, \quad \tau_{i} := \fl(B\tau_{i}), \quad k = 1, \ldots, m_{i}$$

and continue the computations using $p_{k}^{(\nu,i)}$ instead of $q_{k}^{(\nu,i)}$.

6. Numerical Examples

This section presents some results of the numerical experiments showing the performance of Algorithm 8. The compiler is Microsoft Visual Studio 2019. The personal computer used in the examples has Intel Core i7-10510U (1.80 GHz) with 16 GB main memory. We set /Ox and /fp:precise as the compile options.
First, we compare the computing times of Algorithms 5 and 8. The floating-point filter is applied before executing Algorithm 5. For \( k = 1, \ldots, 1000 \) and \( j \in \{1, \ldots, 1000\} \), we generate the argument \( p^{(i)} \in \mathbb{R}^{1000}, i = 1, \ldots, 1000 \) by

\[
p^{(i)} := \begin{cases} 
(2^{-1}, \ldots, 2^{-k}, S_{\text{min}}, \ldots, S_{\text{min}})^T & (i \neq j) \\
(1, S_{\text{min}}, \ldots, S_{\text{min}})^T & (i = j)
\end{cases}
\]  

(34)

The precision of the floating-point numbers is binary64 (double precision). \( p^{(j)} \) is a MaxVector. If \( k \) increases, \( \min_{i \neq j} \left| \sum_{i=1}^{1000} p^{(i)} - \sum_{i=1}^{1000} p^{(j)} \right| \) becomes smaller such that the number of loops in Algorithm 7 increases. Figure 5 depicts the computing times, which are an average of eight runs. If \( k \) is small, the floating-point filter produces the MaxVector; thus, no difference exists in the computing time between Algorithms 5 and 8. For \( k \geq 53 \), the computational time of Algorithm 5 depends on the vector order because Algorithm 5 compares two vectors in a recursive order. On the contrary, the computational time of Algorithm 8 is independent of \( j \). If \( j = 1 \), \( p^{(1)} \) is the MaxVector. This best case for Algorithms 5 and 8 is comparable to that in Algorithm 5. Algorithm 8 is faster than Algorithm 5 for large \( j \) and \( k \).

![Graphs showing computing times for different values of k and j](image-url)

Figure 5: Results for Algorithm 5 for \( m = 1000, n = 1000 \)

We now show the computational speed of Algorithm 8 when an overflow occurs. We compare the computing times to that of Algorithm 8. For \( k = 1, \ldots, 1000 \) and \( j \in \{1, \ldots, 1000\} \), vectors \( q^{(i)} \in \mathbb{R}^{1000}, i = 1, \ldots, 1000 \) is generated by

\[
q^{(i)} := \begin{cases} 
(2^{-1}, \ldots, 2^{-k}, S_{\text{min}}, \ldots, S_{\text{min}}, F_{\text{max}})^T & (i \neq j) \\
(1, S_{\text{min}}, \ldots, S_{\text{min}}, F_{\text{max}})^T & (i = j)
\end{cases}
\]  

(35)

Then, \( q^{(j)} \) is a MaxVector. Figure 6 shows the difference in the performance with/without handling overflow. Handling an overflow slows the performance down by up to 15 times. Such a slowdown can be avoided by applying the floating-point filter after scaling. For \( k > 53 \), the slowdown ratio is from 1.17 to 6.92.

Finally, we check the computing time for obtaining set (3) for the vectors in (34). To obtain set (3), we find (A) the MaxVector and (B) the MinVectors and compute (C) the
sign of $\sum q$ in (21). We have common computations for MaxVector and MinVector, as introduced in Section 3.3 hence, the time for (3) is 55% of the total time for (A) to (C). The computing time ratio is almost independent of $j$ and $k$ in (35).

7. Conclusion

This study proposed an adaptive numerical method for searching for vectors whose sum of all elements was maximum. This method worked accurately, regardless of the rounding error problems, using the error-free transformation for floating-point vectors. If the problem is well-conditioned, the floating-point filter exactly finds the MaxVectors. We also proposed a strategy for the overflow case. The MaxVectors can be obtained by the proper scaling, even if an overflow occurs in the algorithm.

Acknowledgement

This work was supported by JST/CREST, Development of verified numerical computations for mathematical modeling.

References

[1] ANSI/IEEE Std 754-2008, *IEEE Standard for Floating Point Arithmetic*, IEEE, 2008.

[2] S. M. Rump, T. Ogita, S. Oishi: Accurate floating-point summation part I: Faithful rounding, *SIAM J. Sci. Comput.*, 31:1 (2008), 189–224.

[3] T. Ogita, S. M. Rump, S. Oishi: Accurate sum and dot product, *SIAM J. Sci. Comput.*, 26:6 (2005), 1955–1988.

[4] N. Yamanaka, T. Ogita, S. M. Rump, S. Oishi: A parallel algorithm of accurate dot product, *Parallel Comput.*, 34:6–8 (2008), 392–410.

[5] S. M. Rump: Ultimately fast accurate summation, *SIAM J. Sci. Comput.*, 31:5 (2009), 3466–3502.
[6] J. Demmel, Y. Hida: Accurate and efficient floating point summation, *SIAM J. Sci. Comput.*, 25:4 (2004), 1214–1248.

[7] Y.-K. Zhu, W. B. Hayes: Correct rounding and a hybrid approach to exact floating-point summation, *SIAM J. Sci. Comput.*, 31:4 (2009), 2981–3001.

[8] Y.-K. Zhu, W. B. Hayes: Algorithm 908: online exact summation of floating-point streams, *ACM Trans. Math. Softw.*, 37:3 (2010), 37:1–37:13.

[9] 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.

[10] H. Bronnimann, C. Burnikel, S. Pion: Interval arithmetic yields efficient dynamic filters for computational geometry, *Discret. Appl. Math.*, 9:1-2 (2001), 25–47.

[11] G. Melquiond, S. Pion: Formally certified floating-point filters for homogenous geometric predicates, *Theor. Inf. Appl.*, 41:1 (2007), 57–69.

[12] V. Y. Pan, Y. Yu: Certified computation of the sign of a matrix determinant, In: *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, (1999), 715–724.

[13] J. R. Shewchuk: Adaptive precision floating-point arithmetic and fast robust geometric predicates, *Discret. Comput. Geom.*, 18:3 (1997), 305–363.

[14] K. Ozaki, T. Ogita, F. Bunger, S. Oishi, S. M. Rump: Simple floating-point filters for the two-dimensional orientation problem, *BIT Numer. Math.*, 56:2 (2016), 729–749.

[15] T. J. Dekker: A floating-point technique for extending the available precision, *Numer. Math.*, 18:3 (1971), 224–242.

[16] A. H. Karp, P. Markstein: High-precision division and square root, *ACM Trans. Math. Softw.*, 23 (1997), 561–589.

[17] A. Minamihata, K. Ozaki, T. Ogita, S. Oishi: Improved extraction scheme for accurate floating-point summation, In *The 35th JSST Annual Conference International Conference on Simulation Technology*, Kyoto, Japan.

[18] M. Lange, S. M. Rump: Error estimation for the summation of real numbers with application to floating-point summation, *BIT Numer. Math.*, 57 (2017), 927–941.