

A generalized Kahan-Babuška-Summation-Algorithm

Andreas Klein

April 21, 2005

Abstract

In this article we combine recursive summation techniques with Kahan-Babuška type balancing strategies [1, 7] to get highly accurate summation formulas. An i -th algorithm have only $O(n(\log_2(n)\varepsilon)^{i+1})$ error beyond 1upl and thus allows to sum many millions of numbers with high accuracy. The additional afford is a small multiple of the naive summation. In addition we show that these algorithms could be modified to provide tight upper and lower bounds for use with interval arithmetic.

Keywords: floating-point numbers, rounding errors, summation algorithm, interval arithmetic

MSC: 65G40, 65G30

1 Introduction

The analysis of rounding errors in floating-point arithmetic is an important problem of numeric and computer science. A very good introduction into this topic can be found in "What Every Computer Scientist Should Know about Floating-Point Arithmetic"[4].

In this article we want do deal with the following basic problem:

Given n floating-point numbers x_1, \dots, x_n . Compute their sum

$$S = \sum_{k=1}^n x_k .$$

This is a non trivial problem, since the summation of many floating-point numbers always involves rounding. Indeed several algorithms for floating-point-summation were proposed in the last decades. All proposed algorithms fall into one of three classes.

The simplest possible algorithm uses a FOR-loop to add the numbers.

```

s := x[1]
FOR k := 2 TO n DO
  s := s + x[k]
END DO

```

In the following we assume that every floating-point-addition $x \oplus y$ satisfies the bound $x \oplus y = (x + y)(1 + \varepsilon_{x,y})$ mit $|\varepsilon_{x,y}| \leq \varepsilon$. Under these assumptions we can bound the difference between the exact sum and the sum s computed by the FOR-loop by

$$|s - \sum_{k=1}^n x_k| \leq \varepsilon \left[(n-1)|x_1| + \sum_{i=2}^n (n-i+1)|x_i| \right]$$

(see [13]). If the numbers x_i are positive we should sort them in ascending order to minimize the total rounding-error. If the numbers x_i have different sign a good but not necessarily optimal strategy is to sort the numbers increasing to $|x_i|$ (see [5]).

A significant improvement in comparison with the basic iterative algorithm is achieved by recursive algorithms. In these algorithms all binary summation trees are possible. Thus the general recursive algorithm has the form:

1. Start with a list of n numbers x_1, \dots, x_n .
2. Choose two numbers x and y from this list, remove x and y from the list and insert $x \oplus y$.
3. Repeat until the list contains only one element s . This element is the sum $\sum_{k=1}^n x_k$.

If all x_i have positive sign, the optimal summation tree corresponds to the Huffman-tree; i.e. in step 2 we should always choose x and y to be the smallest available numbers. But if the variables x_i have different sign, the determination of the optimal summation-tree is an NP-hard problem. If all

x_i are of the same magnitude, a balanced summation-tree is a good choice (see [8]).

The algorithms of the third class use a totally different strategy. With additional calculations the algorithm tries to estimate the rounding error and correct it afterwards. These algorithms are known as balancing algorithms. The most popular of these algorithms is the Kahan-Algorithm (see [7] or [4]).

```

s := x[1]
c := 0
FOR k := 2 TO n DO
  y := x[k] - c
  t := s + y
  c := (t - s) - y
  s := t
END DO

```

This algorithm guarantees the following error bound

$$s = \sum_{k=1}^n x_k(1 + \delta_k) + O(n\varepsilon^2)$$

with $|\delta_k| \leq 2\varepsilon$.

All known balancing algorithms share the "iterative" structure with the Kahan-Algorithm. As the main contribution of this paper we present a combination of balancing algorithms with recursive algorithms. This yields summation algorithms with very high precision. In addition we will show how to modify balancing algorithms to guarantee an upper or lower bound, respectively. These modified algorithms are suitable for the use in interval-arithmetic.

2 Requirements on the used floating-point arithmetic

Most analysis of floating point arithmetic can be done with the simple error bound

$$x \oplus y = (x + y)(1 + \varepsilon_{x,y})$$

where $|\varepsilon_{x,y}| \leq \varepsilon$ and $\varepsilon > 0$ depends only on the chosen precision. But in the following we will need a more detailed error estimation. Therefore let \oplus

denote floating-point addition with the "round-to-even" strategy and basis 2 or 3 (an IEEE conform floating point unit satisfies this condition). Under these assumptions we have the following result (see Dekker [3] or [9] Theorem C).

Result 1

For $|u| > |v|$ we have

$$u + v = (u \oplus v) + ((u \ominus (u \oplus v)) \oplus v) , \tag{1}$$

i.e. it is possible to compute the rounding error.

All bounds proven in the following sections will use this result, i.e. the bounds are only valid for floating-point arithmetic with "round-to-even" strategy and base 2 or 3. We will discuss other rounding strategies later in section 5.

3 Balancing Algorithms

In this section we describe the algorithms we want to study in this paper. All proofs can be found in the next section.

Unlike the Kahan-Algorithms the algorithms studied in this section correct the rounding error not in each step but at the end of the computation.

The most basic algorithm of this kind is:

```

s := x[1]
c := 0
FOR i := 2 TO n DO
  t := s + x[i]
  IF ABS(s) >= ABS(x[i]) THEN
    c := c + ((s-t)+x[i])
  ELSE
    c := c + ((x[i]-t)+s)
  END IF
  s := t
END DO
s := s + c;

```

This algorithm is essentially the algorithm suggested by Kahan and Babuška (see [1], [7]). The variant presented above appears first in [11]

where it is called improved Kahan-Babuška Algorithm. We can view this algorithm as a special case of the following (new) Algorithm.

Let $c_k^{(0)} = x_k$ for $1 \leq k \leq n$. Compute $s_k^{(i)}$ and $c_k^{(i)}$ recursively by

$$s_0^{(i)} = c_1^{(i)} \quad \text{for } i \geq 0 \quad (2)$$

$$s_k^{(i)} = s_{k-1}^{(i)} \oplus c_{k+1}^{(i)} \quad \text{for } i \geq 0 \text{ and } 1 \leq k < n - i \quad (3)$$

$$c_k^{(i+1)} = \begin{cases} (s_{k-1}^{(i)} \ominus s_k^{(i)}) \oplus c_{k+1}^{(i)} & \text{if } |s_{k-1}^{(i)}| \geq |c_{k+1}^{(i)}| \\ (c_{k+1}^{(i)} \ominus s_k^{(i)}) \oplus s_{k-1}^{(i)} & \text{otherwise} \end{cases} \quad \text{for } i \geq 0 \text{ and } 1 \leq k < n - i \quad (4)$$

As the result of the i -th order iterative Kahan-Babuška Algorithm we define

$$s = \text{round}(s_{n-i-1}^{(i)} + s_{n-i}^{(i-1)} + \dots + s_{n-2}^{(1)} + s_{n-1}^{(0)})$$

The computation of s has to be done in such a way that there is only one rounding operation at the end, i.e. the total rounding error in the computation of s can be bounded by εs . This can be achieved relatively easy. For $i = 0$ and $i = 1$ this condition is satisfied automatically. For $i = 2$, can use the following Algorithm:

- If s_{n-3}^2 , $s_{n-2}^{(1)}$ and $s_{n-1}^{(0)}$ have the same sign, we sort the variables in ascending order and obtain the list a, b, c with $|a| \leq |b| \leq |c|$.

Compute $s = a + b + c$ with the original Kahan-Babuška Algorithm (order 1).

- If the three values s_{n-3}^2 , $s_{n-2}^{(1)}$ and $s_{n-1}^{(0)}$ have different sign sort them into a list with $a \leq 0 \leq b \leq c$ or $a \geq 0 \geq b \geq c$, respectively.

Compute $s = a + b + c$ with the original Kahan-Babuška Algorithm.

For general i we can either choose relatively slow high-precision arithmetic to calculate the exact value of $s_{n-i-1}^{(i)} + s_{n-i}^{(i-1)} + \dots + s_{n-2}^{(1)} + s_{n-1}^{(0)}$ or we can apply, as shown above for the case $i = 2$, a Kahan-Babuška Algorithm of low order on a suitable sorted list.

Since i is small in comparison to n , the cost of the calculation of

$$s = \text{round}(s_{n-i-1}^{(i)} + s_{n-i}^{(i-1)} + \dots + s_{n-2}^{(1)} + s_{n-1}^{(0)})$$

is negligible. (Typical values would be $i = 2$ and $n = 50,000,000$.)

For illustration we give pseudocode for the second order algorithm.

```

s := 0 ;   cs := 0 ;   ccs := 0
FOR j := 1 TO n DO
  t := s + x[i]
  IF ABS(s) >= ABS(x[i]) THEN
    c := (s-t) + x[i]
  ELSE
    c := (x[i]-t) + s
  END IF
  s := t
  t := cs + c
  IF ABS(cs) >= ABS(c) THEN
    cc := (cs-t) + c
  ELSE
    cc := (c-t) + cs
  END IF
  cs := t
  ccs := ccs + cc
END FOR
RETURN s+cs+ccs // no rounding in between

```

For $i = 0$ we obtain the simple iterative summation algorithm. For $i = 1$ we obtain the improved Kahan-Babuška Algorithm described in [11]. For $i \geq 2$ we get new algorithms. Especially the case $i = 2$ yields a significant improvement over the classical Kahan-Babuška Algorithm for a small additional work. (Recently a similar extension of the Kahan-Babuška Algorithm algorithms was presented in [12]. That algorithm satisfies similar bounds as given in Theorem 1, but apparently it has no recursive pardon. A further difference is that in [12] interval arithmetic is not considered.)

The following theorem gives the error bounds for the iterative Kahan-Babuška Algorithms (the proof follows in the next section).

Theorem 1

The i -th order iterative Kahan-Babuška Algorithm satisfies

$$|s - \sum_{k=1}^n x_k| \leq \varepsilon s + \max_{1 \leq k \leq n} |x_k| C_{n,i}^{(I)}(\varepsilon) \quad (5)$$

where

$$C_{n,i}^{(I)}(\varepsilon) = \varepsilon^{i+1} \left(\frac{1}{(i+2)!} n^{i+2} + O(n^{i+1}) \right) + O_n(\varepsilon^{i+2}) \quad (6)$$

(The "constants" of the second O -term still depends on n as indicated by the index n .)

For the special cases $i = 1$ and $i = 2$ we give the preciser bounds

$$C_{n,1}^{(I)}(\varepsilon) = \left(\frac{1}{6}n^3 + \frac{1}{2}n^2 - \frac{2}{3}n - 2 \right) \varepsilon^2 + \left(\frac{1}{12}n^4 + \frac{1}{3}n^3 - \frac{7}{12}n^2 - \frac{11}{6}n + 2 \right) \varepsilon^3 + O_n(\varepsilon^4) \quad (7)$$

and

$$C_{n,2}^{(I)}(\varepsilon) = \left(\frac{1}{24}n^4 + \frac{1}{12}n^3 - \frac{13}{24}n^2 + \frac{5}{12}n - 2 \right) \varepsilon^3 + O_n(\varepsilon^4). \quad (8)$$

It is possible to combine the idea of the Kahan-Babuška Algorithm with the idea of the recursive algorithms. As in the case of the simple recursive algorithm we must choose a suitable addition tree. We give bounds for two variants. For $i > 1$ we only prove bounds for the i -th order recursive Kahan-Babuška Algorithm that use in each stage a balanced binary addition-tree. For the important case $i = 1$ we discuss a specialized variant that produces a better worst case bound.

To keep the description of this specialized first order algorithm simple we assume $n = 2^m$. We start with

$$s_{0,k} = x_k \quad (9)$$

$$s_{j,k} = s_{j-1,2k-1} \oplus s_{j-1,2k} \quad \text{if } 1 \leq j \leq m \text{ and } 1 \leq k \leq 2^{m-j} \quad (10)$$

$$c_{j,k} = \begin{cases} (s_{j-1,2k-1} \ominus s_{j,k}) \oplus s_{j-1,2k} & \text{if } |s_{j-1,2k-1}| \geq |s_{j-1,2k}| \\ (s_{j-1,2k} \ominus s_{j,k}) \oplus s_{j-1,2k-1} & \text{if } |s_{j-1,2k-1}| < |s_{j-1,2k}| \end{cases} \quad (11)$$

We now use balanced binary trees to calculate the sum s'_j of the balancing values $c_{j,k}$. The sum s' of the values s'_j ($1 \leq j \leq m$) is also calculated over a balanced tree. We define

$$s = s_{m,1} \oplus s' \quad (12)$$

as the result of the first order recursive Kahan-Babuška Algorithm.

The following theorem summarizes our results on the recursive Kahan-Babuška Algorithms.

Theorem 2

The first order recursive Kahan-Babuška Algorithm satisfies:

$$\left|s - \sum_{k=1}^n x_k\right| \leq \varepsilon s + \max_{1 \leq k \leq n} |x_k| C_{n,1}^{(R)}(\varepsilon) \quad (13)$$

with

$$C_{n,1}^{(R)}(\varepsilon) = \varepsilon^2 n \left(\frac{(\log_2(n) - 1)(\log_2(n) - 2)}{2} + \log_2(n) \log_2(\log_2(n)) \right) + O_n(\varepsilon^3). \quad (14)$$

For the general i -th order recursive Kahan-Babuška Algorithm we have the error bound

$$\left|s - \sum_{k=1}^n x_k\right| \leq \varepsilon s + \max_{1 \leq k \leq n} |x_k| C_{n,i}^{(R)}(\varepsilon) \quad (15)$$

with

$$C_{n,i}^{(R)}(\varepsilon) = \varepsilon^{i+1} O(n \log^{i+1}(n)) + O_n(\varepsilon^{i+2}) \quad (16)$$

In comparison with the iterative variant we see, that the first order iterative algorithm is good for $n^2 \ll 1/\varepsilon$ while the recursive variant works even for $n \log_2 n \ll 1/\varepsilon$. Similar considerations are true for higher orders. Therefore recursive algorithms should be used if n is very large ($n \approx 1/\varepsilon$ or even higher).

4 Proof of Theorem 1 and 2

We start with the iterative Kahan-Babuška Algorithm.

By Result 1 we have for binary (or ternary) floating-point arithmetic with the round-to-even strategy

$$s_k^{(i)} + c_k^{(i+1)} = s_{k-1}^{(i)} + c_{k+1}^{(i)}$$

for all $i \geq 0$ and $1 \leq k < n - i$.

From this we conclude

$$\begin{aligned} s_{n-1-i}^{(i)} + \sum_{k=1}^{n-1-i} c_k^{(i+1)} &= \left(s_{n-2-i}^{(i)} + \sum_{k=1}^{n-1-i-1} c_k^{(i+1)} \right) + c_{n-1}^{(i)} \\ &\vdots \\ &= \sum_{k=1}^{n-i} c_k^{(i)} \end{aligned}$$

and hence

$$\sum_{k=1}^n x_k = \sum_{k=1}^{n-i} c_k^{(i)} + s_{n-i}^{(i-1)} + s_{n-i+1}^{(i-2)} + \cdots + s_{n-2}^{(1)} + s_{n-1}^{(0)} .$$

The total rounding error of the i -th order iterative Kahan-Babuška Algorithm comes from the error $|s_{n-i-1}^{(i)} - \sum_{k=1}^{n-i} c_k^{(i)}|$ and the rounding error in the last i additions. Since we compute the exact sum

$$s_{n-i-1}^{(i)} + \cdots + s_{n-1}^{(0)}$$

and round only at the end of the computation, the second error is at most εs .

Since $s_{n-i-1}^{(i)}$ is computed by a simple FOR-loop, we have the following error bound

$$|s_{n-i-1}^{(i)} - \sum_{k=1}^{n-i} c_k^{(i)}| \leq |c_1^{(i)}|((1+\varepsilon)^{n-i-1} - 1) + \sum_{k=2}^{n-i} |c_k^{(i)}|((1+\varepsilon)^{n-i+1-k} - 1) \quad (17)$$

(see [13] page 323 or [5] equation (2.4)).

To get an error bound for the i -th order iterative Kahan-Babuška Algorithm we have to compute bounds for $|c_k^{(i)}|$ and $|s_{n-i-1}^{(i)}|$ in dependence on x_1, \dots, x_n .

First we note that by Result 1 we have

$$|c_k^{(i+1)}| \leq \varepsilon |s_k^{(i)}| ,$$

since $c_k^{(i+1)}$ is the rounding error occurring in the computation

$$s_k^{(i)} = s_{k-1}^{(i)} \oplus c_k^{(i)} .$$

Furthermore

$$s_{k-1}^{(i)} = (((c_1^{(i)} + c_2^{(i)})(1 + \eta_1^{(i)}) + c_3^{(i)})(1 + \eta_2^{(i)}) \dots) + c_k^{(i)}(1 + \eta_k^{(i)})$$

with $|\eta_j^{(i)}| \leq \varepsilon$ for $1 \leq j \leq k$. (The relative error in each summation is $\leq \varepsilon$.)

Beginning with $|c_k^{(0)}| = |x_k^{(0)}| \leq \max_{1 \leq j \leq n} |x_j|$ we get bounds for $s_k^{(i)}$ and $c_k^{(i)}$. We find

$$\begin{aligned} s_{k-1}^{(0)} &\leq \max_{1 \leq j \leq n} |x_j| \left((1 + \varepsilon)^{k-1} + \sum_{j=1}^{k-1} (1 + \varepsilon)^j \right) \\ &= \max_{1 \leq j \leq n} |x_j| \left(\sum_{l=1}^{k-1} \varepsilon^l \left[\binom{k-1}{l} + \binom{k-1}{l} + \dots + \binom{1}{l} \right] \right) \end{aligned}$$

and hence

$$|c_k^{(1)}| = \varepsilon(k+1) + \varepsilon^2 \frac{1}{2}(k-1)(k+3) + \text{higher powers of } \varepsilon. \quad (18)$$

Substituting (18) in (17) yields (7).

Similarly we get for $c_k^{(2)}$:

$$\begin{aligned} s_{k-1}^{(1)} &\leq |c_1^{(1)}|(1 + \varepsilon)^{k-1} + \sum_{j=2}^k |c_j^{(1)}|(1 + \varepsilon)^{k+1-j} \\ &\leq (2\varepsilon + 2(k-1)\varepsilon^2) \\ &\quad + \sum_{j=2}^k \left[(j+1)\varepsilon + (j+1)(k+1-j)\varepsilon^2 + \frac{1}{2}(j-1)(j+3)\varepsilon^2 \right] \\ &\quad + \text{higher powers of } \varepsilon \end{aligned} \quad (19)$$

and hence

$$c_k^{(2)} \leq \left(\frac{(k+1)(k+2)}{2} - 1 \right) \varepsilon^2 + \left(\frac{1}{3}k^3 + \frac{7}{4}k^2 - \frac{1}{12}k - 2 \right) \varepsilon^3 + \text{higher powers } \varepsilon.$$

Substituting this in (17) yields (8).

For general i we get by induction

$$\begin{aligned}
|c_k^{i+1}| &\leq \varepsilon |s_{k-1}^{(i)}| \leq \varepsilon \sum_{j=1}^k |c_j| (1 + \varepsilon)^{k+1-j} \\
&\leq \varepsilon \max_{1 \leq l \leq n} |x_l| \left(\sum_{j=1}^k \left(\frac{1}{j!} k^j + O(k^{j-1}) \right) \varepsilon^j + \text{higher powers of } \varepsilon \right) \\
&\hspace{15em} \text{by induction hypothesis} \\
&\leq \max_{1 \leq l \leq n} |x_l| \left(\varepsilon^{i+1} \left(\frac{1}{(i+1)!} k^{i+1} + O(k^i) \right) + \text{higher powers of } \varepsilon \right) .
\end{aligned} \tag{20}$$

Together with (17) this yields (6).

(This analysis also shows that, analogous to the simple iterative algorithm, the iterative Kahan-Babuška Algorithm can be improved by sorting the $c_k^{(i)}$. But for $i \geq 1$ our analysis shows that the $c_k^{(i)}$ are automatically sorted by the worst case bound. Thus in general sorting will give us only a small improvement, that does not justify the additional operations.)

Now we proceed with the analysis of the recursive variants of Kahan-Babuška Algorithm.

First we look at the specialized first order algorithm. According to Result 1 we have

$$|c_{j,k}| \leq \varepsilon |s_{j,k}| \leq 2^j \max_{1 \leq i \leq n} |x_i| .$$

We see that the binary addition tree that we use for the calculation of s' is a Huffman-tree for the worst case bounds of the $c_{j,k}$. As in the case of the iterative Kahan-Babuška Algorithm, Result 1 yields

$$\sum_{k=1}^n x_k = s_{m,1} + \sum_{\substack{1 \leq j \leq m \\ 1 \leq k \leq 2^{m-j}}} c_{j,k} .$$

Thus the error in the calculation of $\sum_{k=1}^n x_k$ comes from the error in the calculation of s' and the error in the summation $s = s_{m,1} \oplus s'$. Since the balanced tree we use for the calculation of s'_j has the height $m - j$, we get the bound

$$\left| s'_j - \sum_{k=1}^{2^{m-j}} c_{j,k} \right| \leq ((1 + \varepsilon)^{m-j} - 1) \sum_{k=1}^{2^{m-j}} |c_{j,k}|$$

(see [10] or [5] equation (3.6)). From this inequality we derivate the bound

$$\begin{aligned}
|s' - \sum_{\substack{1 \leq j \leq m \\ 1 \leq k \leq 2^{m-j}}} c_{j,k}| &\leq \sum_{j=1}^m ((1 + \varepsilon)^{m-j+\log_2(m)} - 1) \sum_{k=1}^{2^{m-j}} |c_{j,k}| \\
&\leq \max_{1 \leq k \leq n} |x_k| \left(\sum_{j=1}^m ((1 + \varepsilon)^{m-j+\log_2(m)} - 1) \sum_{k=1}^{2^{m-j}} \varepsilon 2^j \right) \\
&\leq \max_{1 \leq k \leq n} |x_k| \left(\varepsilon^2 \left(\frac{m}{m-1} 2 + m \log_2(m) - m \right) 2^m \right. \\
&\quad \left. + \text{higher powers of } \varepsilon \right) \\
&= \max_{1 \leq k \leq n} |x_k| \left(\varepsilon^2 n \left(\frac{(\log_2(n) - 2)(\log_2(n) - 1)}{2} \right. \right. \\
&\quad \left. \left. + \log_2(n) \log_2(\log_2(n)) \right) \right) + \text{higher powers of } \varepsilon .
\end{aligned}$$

In general we have in the i -th step the numbers $c_k^{(i-1)}$ and compute their sum by a balanced binary tree. The intermediate sums are denoted by $s_k^{(i-1)}$ and the corresponding balancing values by $c_k^{(i)}$. We have $c_k^{(i)} \leq \varepsilon s_k^{(i-1)}$. Since all intermediate sums are generated by a balanced binary tree, we have

$$\sum_{k=1}^{n-i} s_k^{(i-1)} \leq \lceil \log(n) \rceil \sum_{k=1}^{n-i+1} c_k^{(i-1)} .$$

Thus we find

$$\begin{aligned}
\sum_{k=1}^{n-i} c_k^{(i)} &\leq \varepsilon \lceil \log(n) \rceil \sum_{k=1}^{n-i+1} c_k^{(i-1)} \\
&\leq \varepsilon^2 \lceil \log(n) \rceil^2 \sum_{k=1}^{n-i+2} c_k^{(i-2)} \\
&\leq \dots \\
&\leq \varepsilon^{i-1} \lceil \log(n) \rceil^i \sum_{k=1}^{n-1} c_k^{(i)} \\
&\leq \varepsilon^i \lceil \log(n) \rceil^i \sum_{k=1}^n x_k \\
&\leq n \varepsilon^i \lceil \log(n) \rceil^i \max_{1 \leq k \leq n} |x_k| .
\end{aligned} \tag{21}$$

Since the error of the i -th order algorithm comes from the error in the calculation of $\sum_{k=1}^{n-i} c_k^{(i)}$ and the error in the last i additions, (21) implies (16).

5 Interval arithmetic

For interval arithmetic we need guaranteed upper and lower bounds. The IEEE standard for floating-point arithmetic defines for this reason in addition to the normal "round-to-even" strategy the rounding strategies always "round-down" and always "round-up".

We notate $x \nabla y$ for an addition with the "round-down" strategy and $x \triangle y$ for an addition with the "round-up" strategy.

In the following we will show how to modify the different balancing algorithms to get a guaranteed upper or lower bound, respectively.

We will prove that the following improved version of the Kahan-Summation Algorithm yields upper or lower bounds if we use the round-up or round-down strategy, respectively. For the calculation of the lower bound we use the following variant of the Kahan Algorithm.

$$s_1 = x_1 \quad c_1 = 0 \tag{22}$$

$$y_{k+1} = x_{k+1} \nabla c_k \tag{23}$$

$$s_{k+1} = s_k \oplus y_{k+1} \tag{24}$$

$$c_{k+1} = \begin{cases} (s_{k+1} \ominus s_k) \ominus y_{k+1} & \text{if } |s_k| > |y_{k+1}| \\ (s_{k+1} \ominus y_{k+1}) \ominus s_k & \text{otherwise} \end{cases} \tag{25}$$

$$s = s_n \nabla c_n . \tag{26}$$

Note that a large part of the calculation is done with "normal" round-to-even strategy. (If we want to get an upper bound, we must replace the ∇ by the \triangle operation.)

We prove

Theorem 3

The variant of the Kahan Algorithm defined above yields a guaranteed lower or upper bound, respectively. It satisfies the same error bounds as the original Kahan Algorithm.

Proof

We prove by induction that $s_k - c_k \leq \sum_{j=1}^k x_j$.

For $k = 1$ this is obvious.

For $k \geq 1$ we find

$$\begin{aligned}
s_{k+1} - c_{k+1} &= s_k + y_k && \text{according to Result 1} \\
&= s_k + (x_{k+1} \nabla c_k) \\
&\leq s_k + (x_{k+1} - c_k) \\
&\leq x_{k+1} + \sum_{j=1}^k x_j && \text{by induction hypothesis} \\
&= \sum_{j=1}^{k+1} x_j .
\end{aligned}$$

Thus $s = s_n \nabla c_n \leq s_n - c_n \leq \sum_{j=1}^n x_j$. The error bound of the Kahan Algorithm was proved without an assumption on the rounding strategy and therefore they are also valid for the modified version which use sometimes ∇ instead of \ominus . \square

The adaption of different Kahan-Babuška Algorithms to interval arithmetic is even simpler. We must only use the round-down or round-up strategy for the calculation of $s_{n-1-i}^{(i)}$ and in the calculation of

$$s = \text{round}(s_{n-i-1}^{(i)} + s_{n-i}^{(i-1)} + \dots + s_{n-2}^{(1)} + s_{n-1}^{(0)}) .$$

All other calculation are still down with the "normal" round-to-even strategy. For example the second order iterative Kahan-Babuška Algorithm is described by

$$\begin{aligned}
s_0^{(0)} &= x_1 && s_k^{(0)} = s_{k-1}^{(0)} \oplus x_{k+1} \\
c_k^{(1)} &= \begin{cases} (s_{k-1}^{(0)} \ominus s_k^{(0)}) \oplus x_{k+1} & \text{if } |s_{k-1}^{(0)}| \geq |x_{k+1}| \\ (x_{k+1} \ominus s_k^{(0)}) \oplus s_{k-1}^{(0)} & \text{otherwise} \end{cases} \\
s_0^{(1)} &= c_1^{(1)} && s_k^{(1)} = s_{k-1}^{(1)} \oplus c_{k+1}^{(1)} \\
c_k^{(2)} &= \begin{cases} (s_{k-1}^{(1)} \ominus s_k^{(1)}) \oplus c_{k+1}^{(1)} & \text{if } |s_{k-1}^{(1)}| \geq |c_{k+1}^{(1)}| \\ (c_{k+1}^{(1)} \ominus s_k^{(1)}) \oplus s_{k-1}^{(1)} & \text{otherwise} \end{cases} \\
s_0^{(2)} &= c_1^{(2)} && s_k^{(2)} = s_{k-1}^{(2)} \nabla c_{k+1}^{(2)} \\
s &= \text{round_down}(s_{n-3}^{(2)} + s_{n-2}^{(1)} + s_{n-1}^{(0)}) .
\end{aligned}$$

This yields a guaranteed lower bound.

Theorem 4

The variants of Kahan-Babuška Algorithms described above yield a guaranteed lower or upper bound, respectively. The error bounds of Theorem 1 and 2 are still valid.

Proof

In the following we investigate an i -th order Kahan-Babuška Algorithm. Since we use the round-to-even strategy for calculation of $c_k^{(i)}$, the equation

$$\sum_{k=1}^n x_k = \sum_{k=1}^{n-i} c_k^{(i)} + s_{n-i}^{(i-1)} + s_{n-i+1}^{(i-2)} + \cdots + s_{n-2}^{(1)} + s_{n-1}^{(0)} .$$

is still valid.

We use the round-down or round-up strategy, respectively, for the calculation of $s_{n-i-1}^{(i)}$. Thus we have

$$s_{n-i-1}^{(i)} \leq \sum_{k=1}^{n-i} c_k^{(i)} \quad \text{or} \quad s_{n-i-1}^{(i)} \geq \sum_{k=1}^{n-i} c_k^{(i)} \quad , \text{ respectively.}$$

This proves that the modified Kahan-Babuška Algorithms yields a lower or upper bound for $\sum_{k=1}^n x_k$, respectively.

The error bounds used in the proofs of the Theorems 1 and 2 for $\sum_{k=1}^{n-i} c_k^{(i)}$ do not depend on the exact rounding strategy. (We have only used $x +_{fl} y = (x+y)(1+\varepsilon_{x,y})$ with $\varepsilon_{x,y} \leq \varepsilon$.) Thus the modified Kahan-Babuška Algorithms satisfies the error bound of Theorem 1 and 2. \square

We should note that on some hardware platforms the change of the rounding strategy is a very slow operation. In these cases we should prefer a Kahan-Babuška Algorithm over the Kahan Algorithm and store the $c_k^{(i)}$ in the memory first. By this strategy we can ensure that we need only one change of the rounding strategy during the calculation. This may result in a significant speed up.

6 Computer experiments

We close this article with some computer experiments to quality and speed of the discussed algorithms. All computations where done on a Pentium 2 with 350Mhz, but the relative speed values also hold for other systems. I

used single precision (data-type float) for the computations. For comparison the "exact" sum was calculated with double precision (data-type double).

The summation of 50,000,000 positive numbers of the same magnitude (equally distributed in the intervall [0, 1]) yields:

```

exact sum          : 0x1.7d75ce772217600p+24

iterative          : 0x1.000000000000000p+24      time : 2
recursive          : 0x1.7d75d0000000000p+24      time : 7
Kahan              : 0x1.7d75ce000000000p+24      time : 5
iter. K.-B. order 1 : 0x1.7d765d000000000p+24      time : 5
iter. K.-B. order 2 : 0x1.7d75ce757f00000p+24      time : 7
rec. K.-B. order 1  : 0x1.7d75ce772210000p+24      time : 13

```

As we can see, the recursive Kahan-Babuška Algorithm yields the best result. But the additional effort for second order recursive Kahan-Babuška Algorithm is not worthwhile. The time measure for recursive algorithms should be handled with some care. I used simple recursive C-functions to implement these algorithms. This leads to unnecessary stack operations. The recursive functions can be speeded up if we implement them directly in machine language (see [2] for such techniques). Another advantage of the recursive Algorithms is that they are parallisible (see for example [6] chapter 5.2.2).

If the numbers have different sign we get similar results:

```

exact sum          : 0x1.99189f3dcb00000p+11

iterative          : 0x1.990cb4000000000p+11      time : 2
recursive          : 0x1.99189a000000000p+11      time : 8
Kahan              : 0x1.9918a0000000000p+11      time : 4
iter. K.-B. order 1 : 0x1.99189f478000000p+11      time : 4
iter. K.-B. order 2 : 0x1.99189f3dc500000p+11      time : 7
rec. K.-B. order 1  : 0x1.99189f3dcb00000p+11      time : 13

```

The calculation of upper and lower bounds is more difficult, since under the "round-up" or "round-down" strategy rounding errors can not cancel each other. Under these circumstances the advantages of the Kahan-Babuška Algorithms become even more obvious.

exact sum	: 0x1.7d8a9405b9f3400p+24	
iterative	: 0x1.0000020000000000p+23	time : 2
recursive	: 0x1.7d8a860000000000p+24	time : 7
Kahan	: 0x1.7d8a920000000000p+24	time : 11
iter. K.-B. order 1	: 0x1.57f6020000000000p+24	time : 4
iter. K.-B. order 2	: 0x1.7d89f40000000000p+24	time : 7
rec. K.-B. order 1	: 0x1.7d8a940000000000p+24	time : 13

On a PC the change between the different rounding strategies is very time consuming and should be avoided. For the Kahan-Babuška Algorithms we can achieve this by storing the $c_k^{(i)}$ in memory and do the addition of these numbers at the end. But in the Kahan Algorithm we can not avoid the continuous change of the rounding strategy. This leads to a significant slowdown of this algorithm.

To summarize our results one can say:

Iterative algorithms are only practicable for use with the "normal" (round-to-even) rounding strategy. In this case even the normal Kahan-Babuška Algorithm yields good results. Even higher precision can be achieved by the second order iterative Kahan-Babuška Algorithm or the first order recursive Kahan-Babuška Algorithm.

For the calculation of an upper or lower bound the iterative Kahan-Babuška Algorithms are not suitable. In this case the recursive summation and the Kahan Algorithm yield approximately the same result. But the latter one should be avoided due to the repeated change of the rounding strategy. Further improvement can be achieved by the first order recursive Kahan-Babuška summation.

References

- [1] I. Babuška. Numerical stability in mathematical analysis. *Information Processing*, 68:11–23, 1969.
- [2] O. Caprani. Implementation of a low round-off summation method. *BIT*, 11:271–275, 1971.
- [3] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

- [4] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, 1991. see also: <http://cch.loria.fr/documentation/IEEE754/>.
- [5] N. J. Higham. The accuracy of floating point summation. *SIAM J. Sci. Comput.*, 14(4):783–799, 1993.
- [6] R. W. Hockney and C. R. Jesshope. *Parallel Computers: Architecture, Programming and Algorithms*. Adam Hilger, Bristol, 1981.
- [7] W. Kahan. Further remarks on reducing truncation errors. *Comm. ACM*, 8:40, 1965.
- [8] M.-Y. Kao and J. Wang. Linear-time approximation algorithms for computing numerical summation with provably small errors. *SIAM J. Comput.*, 29(5):1568–1576, 2000.
- [9] D. E. Knuth. *The art of Computer Programming*, volume 2 Seminumerical Algorithms. Addison-Wesley Publishing Company, 1968.
- [10] P. Linz. Accurate floating-point summation. *Comm. ACM*, 13:361–362, 1970.
- [11] A. Neumaier. Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. *Z. angew. Math. Mechanik*, 54:39–51, 1974.
- [12] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot products. *SIAM*, 2005. to appear.
- [13] J. H. Wilkinson. Error analysis of floating-point computation. *Numer. Math.*, 2:319–340, 1960.