

ERROR ESTIMATION OF FLOATING-POINT SUMMATION AND DOT PRODUCT

SIEGFRIED M. RUMP *

Abstract. We improve the well-known Wilkinson-type estimates for the error of standard floating-point recursive summation and dot product by up to a factor 2. The bounds are valid when computed in rounding to nearest. For summation there is no restriction on the number of summands. The proofs are short by using a new tool for the estimation of errors in floating-point computations which cures drawbacks of the “unit in the last place (ulp)”. The presented estimates are nice and simple, and closer to what one may expect.

Key words. floating-point summation, rounding, dot product, unit in the first place (ufp), unit in the last place (ulp), error analysis, error bounds

AMS subject classifications. 15-04, 65G99, 65-04

1. Introduction and Notation. Let $\mathbb{F}$ denote a set of binary floating-point numbers according to the IEEE 754 floating-point standard [3, 4]. Throughout the paper we assume that no overflow occurs, but allow underflow.

The relative rounding error unit, the distance from 1.0 to the next smaller¹ floating-point number, is denoted by **eps**, and the underflow unit by **eta**, that is the smallest positive (subnormal) floating-point number. For IEEE 754 double precision (binary64) we have **eps** = 2^{-53} and **eta** = 2^{-1074} .

We denote by $\text{fl} : \mathbb{R} \rightarrow \mathbb{F}$ a rounding to nearest, that is

$$(1.1) \quad x \in \mathbb{R} : |\text{fl}(x) - x| = \min\{|f - x| : f \in \mathbb{F}\} .$$

Any rounding of the tie can be used without jeopardizing the following estimates, only (1.1) must hold true. This implies that for rounding downwards or upwards or towards zero, all bounds are true *mutatis mutandis* using **2eps** instead of **eps**. Particularly this may be useful for cell processors [5, 6].

For $a, b \in \mathbb{F}$ and $\circ \in \{+, -, \cdot, /\}$, the IEEE 754 floating-point standard defines $\text{fl}(a \circ b) \in \mathbb{F}$ to be the floating-point approximation of $a \circ b \in \mathbb{R}$.

The standard error estimate (see, e.g., [2]) for floating-point addition and subtraction is

$$(1.2) \quad \text{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2) \quad \text{for } a, b \in \mathbb{F}, \circ \in \{+, -\} \quad \text{and} \quad |\varepsilon_1|, |\varepsilon_2| \leq \mathbf{eps} .$$

Note that addition and subtraction is exact near underflow [1], so no underflow unit is necessary in (1.2). More precisely,

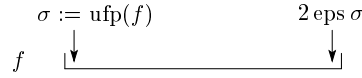
$$(1.3) \quad a, b \in \mathbb{F} \quad \Rightarrow \quad |\text{fl}(a \pm b)| < \mathbf{eps}^{-1} \mathbf{eta} \quad \Rightarrow \quad \text{fl}(a \pm b) = a \pm b .$$

For multiplication and division care is necessary for results in the underflow range:

$$(1.4) \quad \text{fl}(a \circ b) = (a \circ b)(1 + \varepsilon) + \eta \quad \text{for } a, b \in \mathbb{F}, \circ \in \{\cdot, /\} \quad \text{and} \quad |\varepsilon| \leq \mathbf{eps}, |\eta| \leq \mathbf{eta}/2, \varepsilon\eta = 0 .$$

*Institute for Reliable Computing, Hamburg University of Technology, Schwarzenbergstraße 95, Hamburg 21071, Germany, and Visiting Professor at Waseda University, Faculty of Science and Engineering, 3-4-1 Okubo, Shinjuku-ku, Tokyo 169-8555, Japan (rump@tu-harburg.de). Part of this research was done while being visiting Professor at Université Pierre et Marie Curie (Paris 6), Laboratoire LIP6, Département Calcul Scientifique, 4 place Jussieu, 75252 Paris cedex 05, France.

¹Note that sometimes the distance from 1.0 to the next *larger* floating-point number is used; for example, Matlab adopts this rule.

FIG. 1.1. *Normalized floating-point number: unit in the first place and unit in the last place*

For our purposes we need sharper error estimates. In numerical error analysis the “unit in the last place (ulp)” is often used. This concept has several drawbacks: it depends on the floating-point format, extra care is necessary in the underflow range, and it does not apply to real numbers.

Therefore I introduced in [9] the “unit in the first place” (ulp) or leading binary bit of a real number, which is defined by

$$(1.5) \quad 0 \neq r \in \mathbb{R} \quad \Rightarrow \quad \text{ulp}(r) := 2^{\lfloor \log_2 |r| \rfloor} ,$$

where $\text{ulp}(0) := 0$. This concept is independent of a floating point format or underflow range, and it applies to real numbers. It gives a convenient way to characterize the bits of a normalized floating-point number f : they range between the leading bit $\text{ulp}(f)$ and the unit in the last place $2\text{eps} \cdot \text{ulp}(f)$. In particular $\mathbb{F} \subset \text{eta}\mathbb{Z}$, also in underflow. The situation is depicted in Figure 1.1.

Many interesting properties of $\text{ulp}(\cdot)$ are given in [9] without which certain delicate estimations of errors in floating-point computations had not been possible. We need in the following only a few properties which are easily verified.

$$(1.6) \quad 0 \neq x \in \mathbb{R} : \quad \text{ulp}(x) \leq |x| < 2\text{ulp}(x)$$

$$(1.7) \quad x \in \mathbb{R}, f \in \mathbb{F} : \quad |x - f| < \text{eps} \cdot \text{ulp}(x) \quad \Rightarrow \quad \text{fl}(x) = f$$

$$(1.8) \quad r \in \mathbb{R} \quad \Rightarrow \quad \text{ulp}(r) \leq \text{ulp}(\text{fl}(r)) .$$

When rounding $x \in \mathbb{R}$ into $f := \text{fl}(x) \in \mathbb{F}$, the error is sharply characterized by

$$(1.9) \quad f = x + \delta + \eta \quad \text{with} \quad |\delta| \leq \text{eps} \cdot \text{ulp}(x) \leq \text{eps} \cdot \text{ulp}(f) \leq \text{eps}|f|, \quad |\eta| \leq \text{eta}/2, \quad \delta\eta = 0 .$$

This implies in particular for floating-point addition and multiplication

$$(1.10) \quad f = \text{fl}(a + b) \quad \Rightarrow \quad f = a + b + \delta \quad \text{with} \quad |\delta| \leq \text{eps} \cdot \text{ulp}(a + b) \leq \text{eps} \cdot \text{ulp}(f) ,$$

$$(1.11) \quad f = \text{fl}(a \cdot b) \quad \Rightarrow \quad f = a \cdot b + \delta + \eta \quad \text{with} \quad |\delta| \leq \text{eps} \cdot \text{ulp}(a \cdot b) \leq \text{eps} \cdot \text{ulp}(f), \quad |\eta| \leq \text{eta}/2 ,$$

where $\delta\eta = 0$ in the latter case.

2. Summation. In this section let n floating-point numbers $p_i \in \mathbb{F}$ be given. The standard recursive summation computes an approximation of the exact sum $s := \sum_{i=1}^n p_i$ as follows.

ALGORITHM 2.1. *Recursive summation of a vector p_i of floating-point numbers.*

$$\begin{aligned} \tilde{s}_1 &= p_1 \\ \text{for } k &= 2 : n \\ s_k &= \tilde{s}_{k-1} + p_k \\ \tilde{s}_k &= \text{fl}(s_k) \end{aligned}$$

For $n\text{eps} \leq 1$, the computed approximation $\tilde{s}_n$ satisfies the well-known error estimate (cf. [2], Lemma 8.4)

$$(2.1) \quad |\tilde{s}_n - \sum_{i=1}^n p_i| \leq \gamma_{n-1} \sum_{i=1}^n |p_i| \quad \text{with} \quad \gamma_k := \frac{k\text{eps}}{1 - k\text{eps}} ,$$

which follows by recursively applying (1.2). The quantity γ_{n-1} takes care of accumulated rounding errors. Although this is a mathematically clean principle, it lacks computational simplicity. To receive a computable estimation, it gets even worse:

ALGORITHM 2.2. *Recursive summation with error estimation.*

$$\begin{aligned} \tilde{s}_1 &= p_1; \quad \tilde{S}_1 = |p_1| \\ \text{for } k &= 2 : n \\ s_k &= \tilde{s}_{k-1} + p_k; \quad \tilde{s}_k = \text{fl}(s_k) \\ S_k &= \tilde{S}_{k-1} + |p_k|; \quad \tilde{S}_k = \text{fl}(S_k) \end{aligned}$$

The estimate (2.1) can be applied to the floating-point summation of absolute values as well because $|\mathbb{F}| = -|\mathbb{F}|$ implies that taking an absolute value does not cause a rounding error. So abbreviating $S := \sum_{i=1}^n |p_i|$ and applying (2.1) to $\sum |p_i|$ yields

$$(2.2) \quad |\tilde{s}_n - \sum_{i=1}^n p_i| \leq \gamma_{n-1} S \leq \gamma_{n-1} [|\tilde{S}_n| + |\tilde{S}_n - S|] \leq \gamma_{n-1} [|\tilde{S}_n| + \gamma_{n-1} S] ,$$

so that

$$(1 - \gamma_{n-1})S \leq |\tilde{S}_n| .$$

The monotonicity of floating-point operations and $0 \in \mathbb{F}$ gives $|\tilde{S}_n| = \tilde{S}_n$, so that for $2(n-1)\mathbf{eps} < 1$ a little computation proves

$$(2.3) \quad |\tilde{s}_n - \sum_{i=1}^n p_i| \leq \frac{(n-1)\mathbf{eps}}{1 - 2(n-1)\mathbf{eps}} \tilde{S}_n .$$

In order to arrive at a (computable) floating-point number bounding the error, extra care is necessary when evaluating the right hand side of (2.3).

The classical bound (2.1) can be freed from the nasty $\mathcal{O}(\mathbf{eps}^2)$ term. To show this we need the following auxiliary lemma. Note the beauty of the proof using the ufp-concept.

LEMMA 2.3. *Let $a, b \in \mathbb{F}$. Then*

$$(2.4) \quad |\text{fl}(a+b) - (a+b)| \leq |b| .$$

PROOF. If $|(a+b) - a| < \mathbf{eps} \cdot \text{ufp}(a+b)$, then (1.7) implies $\text{fl}(a+b) = a$ and therefore (2.4). If $\text{fl}(a+b) \neq a$, then (1.10) yields

$$|b| = |(a+b) - a| \geq \mathbf{eps} \cdot \text{ufp}(a+b) \geq |\text{fl}(a+b) - (a+b)| . \quad \square$$

THEOREM 2.4. *Let $p_i \in \mathbb{F}$ for $1 \leq i \leq n$, and let $\tilde{s}_n$ be the quantity computed by recursive summation as in Algorithm 2.1. Then*

$$(2.5) \quad |\tilde{s}_n - \sum_{i=1}^n p_i| \leq (n-1)\mathbf{eps} \sum_{i=1}^n |p_i| .$$

REMARK 1. Note that in contrast to (2.1) there is no restriction on n , whereas the classical bound (2.1) becomes very weak for $n\mathbf{eps}$ close to 1.

PROOF. Proceeding by induction we note

$$(2.6) \quad \Delta := \left| \tilde{s}_n - \sum_{i=1}^n p_i \right| = \left| \tilde{s}_n - s_n + \tilde{s}_{n-1} - \sum_{i=1}^{n-1} p_i \right| \leq |\tilde{s}_n - s_n| + (n-2)\mathbf{eps} \sum_{i=1}^{n-1} |p_i| .$$

We distinguish two cases. First, assume $|p_n| \leq \mathbf{eps} \sum_{i=1}^{n-1} |p_i|$. Then Lemma 2.3 implies

$$(2.7) \quad |\tilde{s}_n - s_n| = |\mathbf{fl}(\tilde{s}_{n-1} + p_n) - (\tilde{s}_{n-1} + p_n)| \leq |p_n| \leq \mathbf{eps} \sum_{i=1}^{n-1} |p_i| ,$$

and inserting into (2.6) finishes this part of the proof. Henceforth, assume $\mathbf{eps} \sum_{i=1}^{n-1} |p_i| < |p_n|$. Then (1.9) gives

$$|\tilde{s}_n - s_n| \leq \mathbf{eps}|s_n| = \mathbf{eps}|\tilde{s}_{n-1} - \sum_{i=1}^{n-1} p_i + \sum_{i=1}^n p_i| ,$$

so that (2.6) yields

$$\begin{aligned} \Delta &\leq \mathbf{eps} \left[(n-2)\mathbf{eps} \sum_{i=1}^{n-1} |p_i| + \sum_{i=1}^n |p_i| \right] + (n-2)\mathbf{eps} \sum_{i=1}^{n-1} |p_i| \\ &< \mathbf{eps} \left[(n-2)|p_n| + |p_n| + \sum_{i=1}^{n-1} |p_i| \right] + (n-2)\mathbf{eps} \sum_{i=1}^{n-1} |p_i| \\ &= (n-1)\mathbf{eps} \sum_{i=1}^n |p_i| . \end{aligned}$$

The proof is finished. $\square$

REMARK 2. The estimate (2.5) is almost sharp as for $p_1 = 1$ and $p_i = \mathbf{eps}$ for $1 \leq i \leq n$. There are obvious possibilities to improve it, in particular p_n does not occur in (2.7); however, this would sacrifice its simplicity.

For a computable bound, we may proceed as in (2.2). For $\tilde{S}$ denoting a floating-point approximation of $\sum |p_i|$ computed by Algorithm 2.1 in any order of summation, Theorem 2.4 yields

$$\left| \tilde{s}_n - \sum_{i=1}^n p_i \right| \leq (n-1)\mathbf{eps}S \leq (n-1)\mathbf{eps} \left[|\tilde{S}| + (n-1)\mathbf{eps}S \right] ,$$

and therefore

$$(2.8) \quad \left| \tilde{s}_n - \sum_{i=1}^n p_i \right| \leq \gamma_{n-1} \tilde{S} .$$

However, this estimate still contains the unwieldy $\mathcal{O}(\mathbf{eps}^2)$ term.

To arrive at an easily computable and nice bound, we analyze Algorithm 2.2 further. We will use a monotonicity property of a rounding defined by (1.1), namely

$$(2.9) \quad |x| \leq X \text{ and } |y| \leq Y \Rightarrow |\mathbf{fl}(x+y)| \leq \mathbf{fl}(X+Y) \quad \text{for } x, y, X, Y \in \mathbb{R} ,$$

so that

$$(2.10) \quad |\tilde{s}_k| \leq \tilde{S}_k \quad \text{for } 1 \leq k \leq n$$

for the quantities computed by Algorithm 2.2.

THEOREM 2.5. *Let $p_i \in \mathbb{F}$ for $1 \leq i \leq n$, and let $\tilde{s}_n$ and $\tilde{S}_n$ be the quantities computed by recursive summation as in Algorithm 2.2. Then*

$$(2.11) \quad \left| \tilde{s}_n - \sum_{i=1}^n p_i \right| \leq (n-1)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) \quad \left[\leq (n-1)\mathbf{eps}\tilde{S}_n \right] .$$

The estimation is sharp for all n .

REMARK 3. As for the previous theorem we note that there is no restriction on n .

PROOF. We use (1.9) and (2.10) to see

$$|\tilde{s}_k - s_k| \leq \mathbf{eps} \cdot \mathbf{ufp}(\tilde{s}_k) \leq \mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_k)$$

for $2 \leq k \leq n$, so that by induction

$$\begin{aligned} |\tilde{s}_k - \sum_{i=1}^k p_i| &= |\tilde{s}_k - s_k + \tilde{s}_{k-1} - \sum_{i=1}^{k-1} p_i| \\ &\leq \mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_k) + (k-2)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_{k-1}) \\ &\leq (k-1)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_k). \end{aligned}$$

This proves (2.11). Setting $p_1 := 1$ and $p_i := \mathbf{eps}$ for $1 \leq i \leq n$ we obtain in IEEE 754 rounding tie to even $\mathbf{fl}(1 + \mathbf{eps}) = 1$, so that $\tilde{s}_n = 1 = \tilde{S}_n$. Hence $|\tilde{s}_n - \sum_{i=1}^n p_i| = (n-1)\mathbf{eps} = (n-1)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n)$. This completes the proof. $\square$

The bound in (2.11), unlike (2.3) and (2.8), is only valid when both $\tilde{s}_n$ and $\tilde{S}_n$ are computed in the same order of summation. Concerning a practical application of (2.11), the unit in the first place can be computed with 4 floating-point operations. Algorithm 3.5 in [8], which we repeat for convenience, improves an earlier version by the author by avoiding branches. It works correct in the underflow range but causes overflow for input very close to the largest representable floating-point number. The latter case is cured by scaling.

ALGORITHM 2.6. *Unit in the first place of a floating-point number.*

$$\begin{aligned} &\text{function } S = \mathbf{ufp}(p) \\ &\quad q = \mathbf{fl}(\varphi p) \qquad \qquad \text{for } \varphi := (2\mathbf{eps})^{-1} + 1 \\ &\quad S = \mathbf{fl}(|q - (1 - \mathbf{eps})q|) \end{aligned}$$

In combination with Theorem 2.4 this allows an easy computation of a rigorous error bound for floating-point summation, whereas a rigorous bound based on (2.3) or (2.8) in rounding to nearest is unwieldy.

COROLLARY 2.7. *Let $p_i \in \mathbb{F}$ for $1 \leq i \leq n$, assume $n\mathbf{eps} \leq 1$, and let $\tilde{s}_n, \tilde{S}_n$ and $r := \mathbf{ufp}(\tilde{S}_n)$ be the quantities computed by Algorithms 2.2 and 2.6. Then*

$$(2.12) \quad |\tilde{s}_n - \sum_{i=1}^n p_i| \leq \mathbf{fl}((n-1) \cdot \mathbf{fl}(\mathbf{eps} \cdot r)).$$

The estimation is sharp for all $n \leq \mathbf{eps}^{-1}$.

PROOF. Denote the floating-bound in (2.12) by $\tilde{R}$. If $\tilde{S}_n < \mathbf{eps}^{-1}\mathbf{eta}$, then (2.10) and (1.3) imply that there is no rounding error at all in the summations producing $\tilde{s}_n$ and $\tilde{S}_n$, i.e. $\tilde{s}_n = \sum p_i$. In rounding tie to even also $\mathbf{fl}(\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n)) = \mathbf{fl}(\mathbf{eta}/2) = 0 = \tilde{R}$.

Otherwise, since $\mathbf{ufp}(\tilde{S}_n)$ is a power of 2, $\mathbf{ufp}(\tilde{S}_n) \geq \mathbf{eps}^{-1}\mathbf{eta}$ implies that for n not larger than $\mathbf{eps}^{-1}$ no rounding error occurs in the computation of $\tilde{R}$, so Theorem 2.5 finishes the proof. $\square$

REMARK 4. One might be inclined to improve Theorem 2.4 along the lines of Theorem 2.5 into

$$|\tilde{s}_n - \sum_{i=1}^n p_i| \leq (n-1)\mathbf{eps} \cdot \mathbf{ufp}(\sum_{i=1}^n |p_i|).$$

However, this is not true as by

$$p_{1\dots 5} := [1 - 5\mathbf{eps}, \frac{\mathbf{eps}}{2}, \frac{3}{2}\mathbf{eps}, \frac{3}{2}\mathbf{eps}, \mathbf{eps}(1 + 2\mathbf{eps})].$$

First note that $p_i \in \mathbb{F}$. Second, $\tilde{S}_{2\dots 5} = \tilde{s}_{2\dots 5} = 1 + \mathbf{eps} \cdot [-4, -2, 0, 2]$ and $s := \sum p_i = 1 - \frac{\mathbf{eps}}{2} + 2\mathbf{eps}^2$. Hence

$$| \tilde{s}_n - \sum_{i=1}^n p_i | = | \tilde{s}_5 - s | = | (1 + 2\mathbf{eps}) - (1 - \frac{\mathbf{eps}}{2} + 2\mathbf{eps}^2) | = \frac{5}{2}\mathbf{eps} - 2\mathbf{eps}^2 ,$$

but

$$(n-1)\mathbf{eps} \cdot \mathbf{ufp}(\sum_{i=1}^n |p_i|) = 4\mathbf{eps} \cdot \mathbf{ufp}(1 - \frac{\mathbf{eps}}{2} + 2\mathbf{eps}^2) = 2\mathbf{eps} < \frac{5}{2}\mathbf{eps} - 2\mathbf{eps}^2 .$$

3. Dot product. In this section let $2n$ floating-point numbers $x_i, y_i \in \mathbb{F}$, $1 \leq i \leq n$ be given. The standard recursive routine computes an approximation of the exact dot product $x^T y$ as follows.

ALGORITHM 3.1. *Dot product of two vectors $x = (x_1, \dots, x_n)$, $y = (y_1, \dots, y_n)$ of floating-point numbers.*

$$\begin{aligned} \tilde{p}_1 &= \mathbf{fl}(x_1 \cdot y_1); \tilde{s}_1 = \tilde{p}_1 \\ \text{for } k &= 2 : n \\ p_k &= x_k \cdot y_k; \tilde{p}_k = \mathbf{fl}(p_k) \\ s_k &= \tilde{s}_{k-1} + \tilde{p}_k \\ \tilde{s}_k &= \mathbf{fl}(s_k) \end{aligned}$$

If no underflow occurs and $n\mathbf{eps} < 1$, the computed approximation $\tilde{s}_n$ satisfies the well-known error estimate (cf. [2], Chapter 3, p. 63)

$$(3.1) \quad |\tilde{s}_n - x^T y| \leq \gamma_n |x^T| |y| ,$$

where the absolute value of vectors is taken entrywise. Next we extend this algorithm to obtain a computable error estimate including underflow.

ALGORITHM 3.2. *Dot product of two vectors $x, y \in \mathbb{F}^n$ with error estimate.*

$$\begin{aligned} \tilde{p}_1 &= \mathbf{fl}(x_1 \cdot y_1); \tilde{s}_1 = \tilde{p}_1; \tilde{S}_1 = |\tilde{p}_1| \\ \text{for } k &= 2 : n \\ p_k &= x_k y_k; \tilde{p}_k = \mathbf{fl}(p_k) \\ s_k &= \tilde{s}_{k-1} + \tilde{p}_k; \tilde{s}_k = \mathbf{fl}(s_k) \\ S_k &= \tilde{S}_{k-1} + |\tilde{p}_k|; \tilde{S}_k = \mathbf{fl}(S_k) \end{aligned}$$

Using the results of the previous section we derive the following computable error estimate for the floating-point approximation $\tilde{s}_n$ computed by Algorithm 3.2 to the exact dot product $x^T y$.

THEOREM 3.3. *Let $x, y \in \mathbb{F}^n$ with $(n+2)\mathbf{eps} \leq 1$, and let $\tilde{s}_n$ and $\tilde{S}_n$ be the quantities computed by Algorithm 3.2. Denote by $\mathbf{realmin} := \frac{1}{2}\mathbf{eps}^{-1}\mathbf{eta}$ the smallest positive normalized floating-point number. Then*

$$(3.2) \quad | \tilde{s}_n - x^T y | \leq (n+2)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) + \mathbf{realmin} .$$

The factor $n+2$ cannot be replaced by $n+1$.

REMARK 5. It is not possible to avoid some additive term covering underflow in (3.2) since all $|x_i y_i|$ may be so small that $\tilde{s}_n = \tilde{S}_n = 0$ but $x^T y \neq 0$.

PROOF. Applying Theorem 2.5 to the vector $(|\tilde{p}_1|, \dots, |\tilde{p}_n|) \in \mathbb{F}^n$ and using $n\mathbf{eps} \leq 1$ gives

$$(3.3) \quad | \tilde{S}_n - \sum_{i=1}^n |\tilde{p}_i| | \leq (n-1)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) < \mathbf{ufp}(\tilde{S}_n) ,$$

and again applying it to $\tilde{p}_i$ shows

$$(3.4) \quad \left| \tilde{s}_n - \sum_{i=1}^n p_i \right| \leq (n-1)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) + \left| \sum_{i=1}^n (\tilde{p}_i - p_i) \right|.$$

Next (1.9) yields

$$|\tilde{p}_i - p_i| = |\mathbf{fl}(p_i) - p_i| \leq \mathbf{eps} \cdot |\tilde{p}_i| + \mathbf{eta}/2,$$

so that (3.3) and (1.6) imply

$$\begin{aligned} \left| \sum_{i=1}^n (\tilde{p}_i - p_i) \right| &\leq \mathbf{eps} \sum_{i=1}^n |\tilde{p}_i| + n\mathbf{eta}/2 \\ &< \mathbf{eps} [\tilde{S}_n + \mathbf{ufp}(\tilde{S}_n)] + \mathbf{realmin} \\ &< 3\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) + \mathbf{realmin}. \end{aligned}$$

With (3.4) this proves (3.2). Finally consider the following example showing that the factor $n+2$ cannot be replaced by $n+1$. In a floating-point format with k bits in the mantissa, i.e. $\mathbf{eps} = 2^{-k}$, define

$$(3.5) \quad x_i := \begin{cases} 2^{-1}(1 + 2\mathbf{eps}) & \text{for } i = 1 \\ 2^{-i}(1 + 2\mathbf{eps}) + \frac{\mathbf{eps}}{2} & \text{for } 2 \leq i \leq k \\ \frac{\mathbf{eps}}{2}(1 + 2\mathbf{eps}) & \text{for } k+1 \leq i \leq n \end{cases}$$

and $y_{1\dots n} := 1 - \mathbf{eps}$. Note that $x_i, y_i \in \mathbb{F}$. Applying Algorithm 3.2 yields

$$\tilde{p}_1 = 2^{-1}, \quad \tilde{p}_i := 2^{-i} + \frac{\mathbf{eps}}{2} \quad \text{for } 2 \leq i \leq k \quad \text{and} \quad \tilde{p}_i := \frac{\mathbf{eps}}{2} \quad \text{for } k+1 \leq i \leq n.$$

Therefore

$$\tilde{S}_n = \tilde{s}_n = \sum_{i=1}^n 2^{-i} = 1 - \mathbf{eps} \quad \text{and} \quad \mathbf{ufp}(\tilde{S}_n) = \frac{1}{2}.$$

But

$$\begin{aligned} x^T y &= (1 - \mathbf{eps}) \sum_{i=1}^n x_i \\ &= (1 - \mathbf{eps}) \left[(1 - \mathbf{eps})(1 + 2\mathbf{eps}) + (k-1)\frac{\mathbf{eps}}{2} + (n-k)\frac{\mathbf{eps}}{2}(1 + 2\mathbf{eps}) \right] \\ &= (1 - \mathbf{eps}) \left[1 + (n+1)\frac{\mathbf{eps}}{2} + (n-k-2)\mathbf{eps}^2 \right] \\ &> 1 + (n-1)\frac{\mathbf{eps}}{2} + (n-2k-5)\frac{\mathbf{eps}^2}{2}, \end{aligned}$$

so that

$$x^T y - \tilde{s}_n > 1 + (n+1)\frac{\mathbf{eps}}{2} + (n-2k-5)\frac{\mathbf{eps}^2}{2}.$$

Therefore, in view of $\mathbf{ufp}(\tilde{S}_n) = \frac{1}{2}$, the factor $n+2$ in (3.2) cannot be replaced by $n+1$. In IEEE 754 double precision (binary64) this is (at least) the case for vector lengths larger than 111. This completes the proof. $\square$

The example in the proof shows that for the largest admissible vector length $n = \mathbf{eps}^{-1} - 2$ it may happen that

$$|\tilde{s}_n - x^T y| > \phi \cdot \{ (n+2)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) + \mathbf{realmin} \} \quad \text{with} \quad \phi := 1 - (2k+8)\mathbf{eps}^2.$$

For IEEE 754 double precision (binary64), $\phi > 1 - 10^{-29}$.

Finally we show how to compute in floating-point rounding to nearest an upper bound for the error of the floating-point dot product. The proof is a little tricky because care is necessary for input near or in the underflow range.

COROLLARY 3.4. Let $x, y \in \mathbb{F}^n$ with $(n+2)\mathbf{eps} \leq 1$ be given, denote by $\mathbf{realmin} := \frac{1}{2}\mathbf{eps}^{-1}\mathbf{eta}$ the smallest positive normalized floating-point number, and let $\tilde{s}_n, \tilde{S}_n$ and $r := \text{ufp}(\tilde{S}_n)$ be the quantities computed by Algorithm 3.2. Define $\varrho := \frac{3}{2}\mathbf{realmin}$ and let

$$(3.6) \quad \tilde{R} := \text{float}((n+2) \cdot (\mathbf{eps} \cdot r) + \varrho) \quad \text{and} \quad \hat{R} := \text{float}((n+2) \cdot \mathbf{eps}) \cdot r + \varrho,$$

where $\text{float}(\cdot)$ means that all operations are carried out in floating-point arithmetic rounded to nearest in the specified order. Then

$$(3.7) \quad |\tilde{s}_n - x^T y| \leq \tilde{R} \quad \text{and} \quad |\tilde{s}_n - x^T y| \leq \hat{R}.$$

REMARK 6. Note that $\text{fl}(n+2) = n+2$ and $\text{fl}(\frac{3}{2} \cdot \mathbf{realmin}) = \varrho$.

REMARK 7. The first quantity $\tilde{R}$ in (3.7) is advantageous for $\text{ufp}(\tilde{S}_n)$ near the underflow range, in the underflow range $\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n)$ even becomes zero. When applying Corollary 3.4 to matrix multiplication, however, $\text{ufp}(\tilde{S}_n)$ becomes a matrix and the second quantity $\hat{R}$ is preferable because only one multiplication of a scalar times a matrix is necessary.

PROOF. We distinguish three cases. First, assume $\tilde{S}_n \geq \mathbf{eps}^{-1}\mathbf{realmin}$, so that $\text{ufp}(\tilde{S}_n) \geq \mathbf{eps}^{-1}\mathbf{realmin}$. Then both products in the floating-point computation of $(n+2) \cdot \mathbf{eps} \cdot \text{ufp}(\tilde{S}_n)$ do not cause a rounding error. Therefore

$$\tilde{R} = \hat{R} \geq (n+2)\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n)$$

because $\text{fl}(a+b) \geq a$ for any nonnegative $a, b \in \mathbb{F}$. Hence, in view of (3.4), it suffices to prove

$$(3.8) \quad \left| \sum_{i=1}^n (\tilde{p}_i - p_i) \right| \leq 3\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n)$$

under the assumption $\text{ufp}(\tilde{S}_n) \geq \mathbf{eps}^{-1}\mathbf{realmin}$. Denote by $I \subseteq \{1, \dots, n\}$ the index set for which $|\tilde{p}_i| < \mathbf{realmin}$. Then $\varepsilon\eta = 0$ in (1.9) yields

$$|\tilde{p}_i - p_i| \leq \begin{cases} \mathbf{eta}/2 & \text{for } i \in I \\ \mathbf{eps}|\tilde{p}_i| & \text{otherwise,} \end{cases}$$

so that

$$(3.9) \quad \left| \sum_{i=1}^n (\tilde{p}_i - p_i) \right| \leq \mathbf{eps} \sum_{i \notin I} |\tilde{p}_i| + k\mathbf{eta}/2 \quad \text{for } k := |I|.$$

Denote by $\tilde{S}_n^*$ the result of Algorithm 2.2 when applied to the vector $\{ |\tilde{p}_i| : i \notin I \}$, that is leaving out the vector elements in the index set I . Recursively applying (2.9) implies $\tilde{S}_n^* \leq \tilde{S}_n$ because the order of summation is not changed. Thus Theorem 2.5, (1.6) and $n\mathbf{eps} < 1$ give

$$\sum_{i \notin I} |\tilde{p}_i| \leq \tilde{S}_n^* + (n-k-1)\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n^*) < (2 + (n-k-1)\mathbf{eps}) \cdot \text{ufp}(\tilde{S}_n) < (3 - k\mathbf{eps})\text{ufp}(\tilde{S}_n).$$

Hence (3.9) and $\text{ufp}(\tilde{S}_n) \geq \mathbf{eps}^{-1}\mathbf{realmin}$ yield

$$\left| \sum_{i=1}^n (\tilde{p}_i - p_i) \right| \leq (3 - k\mathbf{eps})\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n) + k\mathbf{eta}/2 \leq 3\mathbf{eps} \cdot \text{ufp}(\tilde{S}_n)$$

and prove (3.8) as desired. This finishes the first case.

Second, assume $\mathbf{eps}^{-1}\mathbf{eta} \leq \tilde{S}_n < \mathbf{eps}^{-1}\mathbf{realmin}$. Then still there is no rounding error in the floating-point computation of $\Phi := (n+2) \cdot \mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n)$, so that $\Phi = \tilde{R} = \hat{R} \leq \mathbf{ufp}(\tilde{S}_n) \leq \frac{1}{2}\mathbf{eps}^{-1}\mathbf{realmin}$ and

$$\mathbf{ufp}(\Phi + \varrho) \leq \mathbf{ufp}\left(\frac{1}{2}\mathbf{eps}^{-1}(1 + 3\mathbf{eps})\mathbf{realmin}\right) = \frac{1}{2}\mathbf{eps}^{-1}\mathbf{realmin}.$$

Now using (1.10) shows

$$\tilde{R} = \hat{R} = \mathbf{fl}(\Phi + \varrho) \geq \Phi + \varrho - \mathbf{eps} \cdot \mathbf{ufp}(\Phi + \varrho) \geq (n+2)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n) + \mathbf{realmin}$$

and Theorem 3.3 applies.

It remains the third case $\tilde{S}_n < \mathbf{eps}^{-1}\mathbf{eta}$. But then by (1.3), as already discussed, there is no rounding error at all in the summations producing $\tilde{s}_n$ and $\tilde{S}_n$. Then, because the result of the floating-point computation of $(n+2)\mathbf{eps} \cdot \mathbf{ufp}(\tilde{S}_n)$ in any order is nonnegative, the total error $|\tilde{s}_n - x^T y|$ is bounded by $n\mathbf{eta}/2 < \mathbf{realmin} < \varrho \leq \min(\tilde{R}, \hat{R})$. The proof is finished. $\square$

We finally note that the way Corollary 3.4 handles the underflow has the computational advantage that a *normalized* number is added to the computed result $\mathbf{ufp}(\tilde{S}_n)$: floating-point arithmetic in the underflow range on today's architectures is extremely slow.

We mention that much faster algorithms are possible if the entries of the input data do not differ too much in absolute value. In this case Ogita and Oishi [7] propose to compute the rowwise and columnwise maximum of the absolute value of the first and second factor, respectively, and bound the entries of the product by their outer product. Then only the product of the midpoints is to be computed whereas eventually the radius and all errors are estimated in $\mathcal{O}(n^2)$.

4. Computational results. Following we first compare the (computable) Wilkinson-type standard estimate (2.3) with the new computable bound (2.12) in Corollary 2.7. For simplicity we compute (2.3) in floating-point ignoring rounding errors. The difference is marginal, though this bound is not necessarily rigorous.

Care is necessary to generate suitable tests for summation. Picking p_i from some random distribution, the mean value for $\sum p_i$ is easily computed and the ratio between the Wilkinson-type to the new bound (2.12) can be predicted. For example, the mean deviation for uniformly distributed input data with mean zero and standard deviation 1 is $\sqrt{\frac{2}{\pi}} =: \mu \approx 0.7979$, so that the expected value for $\sum |p_i|$ is $n\mu$. For $n = 10^4$ this means $\sum |p_i| \approx 7979$ and probably $\mathbf{ufp}(\tilde{S}_n) = 4096$, so that the ratio between (2.3) and (2.12) will probably be almost 2.

To arrive at sufficiently random input data, we use vectors $\mathbf{p} = \mathbf{randn}(1, K) * (\mathbf{rand}(K) * \mathbf{randn}(K, n))$ in Matlab notation for $K = 1000$. Such vectors should contain a certain randomness reflecting the behavior of floating-point rounding - although (fortunately) such rounding is by no means random [10]. For 1000 tests each the minimum, mean, median and maximum ratio between the estimates (2.3) with (2.12) is displayed in Table 4.1.

As expected, the ratio is between 1 and 2, so that the new estimate is up to a factor 2 better than the traditional one. On the average, the old bound seems to exceed the new one by about 40%.

Comparing the standard Wilkinson-type estimate for dot products (3.1) with the new computable one (3.6) in Corollary 3.4 we generate a matrix A with anticipated condition number 10^k by specifying the singular values geometrically decreasing from 1 to 10^{-k} , and pre- and postmultiplying a randomly chosen orthogonal matrix. This should generate a matrix with suitably random entries. We compute an approximate inverse R of A in floating-point and estimate the error in the computation of RA .

The results are displayed in Table 4.2, where the Wilkinson-type bound (3.1) is computed in floating-point ignoring rounding errors. We did not observe a significant difference for varying condition numbers, so it is

TABLE 4.1

Minimum, mean, median and maximum ratio between the estimates (2.3) with (2.12).

n	minimum	mean	median	maximum
10	1.0002	1.4338	1.3976	1.9982
100	1.0009	1.4767	1.4727	1.9960
1000	1.0007	1.4611	1.4201	1.9960
10,000	1.0004	1.3795	1.3086	1.9987
100,000	1.0033	1.4713	1.4508	1.9963

TABLE 4.2

Minimum, mean, median and maximum ratio between the estimates (3.1) with (3.6).

n	minimum	mean	median	maximum
50	0.9618	1.3967	1.3760	1.9212
100	0.9805	1.4133	1.3869	1.9606
200	0.9901	1.4574	1.4562	1.9801
500	0.9960	1.5630	1.5840	1.9920
1000	0.9980	1.5374	1.5893	1.9960

fixed to 10^{12} . As can be seen we can expect an improvement by a factor 1.4, as previously for summation. Due to the factor $n + 2$ in (3.6) compared to $\gamma_n \approx n$ in (3.1) the new estimate for dot products may also be worse than the traditional one. This could be fixed by taking the minimum.

5. Conclusion. The new bounds are nice by omitting higher order terms, and close to what would expect (or desire). Moreover they are easy to compute in floating-point rounding to nearest and improve the classical Wilkinson-type bounds by up to a factor of 2. As is well-known the worst case is very rare, so that both types of bounds usually grossly overestimate the true error. Nevertheless the new bounds fill some theoretical gap, they demonstrate the power of the ufp-concept, and they do not require directed rounding.

Despite the theoretical merit the bounds may be useful in applications where rigorous bounds are mandatory and directed rounding is not available. For example in computer-assisted proofs or interval libraries for scalar, vector and matrix operations.

For $\tilde{s}_n$ denoting the result of Algorithm 3.2 we conjecture

$$(5.1) \quad |\tilde{s}_n - x^T y| \leq \text{neps} |x^T| |y| + \text{realmin}$$

for $\text{neps} \leq 1$, but could neither prove it nor find a counterexample.

Acknowledgement. My thanks to Florian Bünger, Stef Graillat and Takeshi Ogita for helpful comments.

REFERENCES

- [1] J.R. Hauser. Handling floating-point exceptions in numeric programs. *ACM Trans. Program. Lang. Syst.*, 18(2):139–174, 1996.
- [2] N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM Publications, Philadelphia, 2nd edition, 2002.
- [3] *ANSI/IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic*. New York, 1985.
- [4] *ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic*. New York, 2008.
- [5] C. Jacobi, H.J. Oh, K.D. Tran, S.R. Cottier, B.W. Michael, H. Nishikawa, Y. Totsuka, T. Namatame, and N. Yano. The vector floating-point unit in a synergistic processor element of a cell processor. In *ARITH 2005: Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, pages 59–67, Washington, 2005.

- [6] J.M. Muller, N. Brisebarre, F. de Dinechin, C.P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.
- [7] T. Ogita and S. Oishi. Fast Inclusion of Interval Matrix Multiplication. *Reliable Computing*, 11:191–205, 2005.
- [8] S.M. Rump. Ultimately Fast Accurate Summation. *SIAM Journal on Scientific Computing (SISC)*, 31(5):3466–3502, 2009.
- [9] S.M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.*, 31(1):189–224, 2008.
- [10] L. N. Trefethen and R. Schreiber. Average-case stability of gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(3):335–360, 1990.