accepted for publication in BIT, May 18, 2011.

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, no higher order terms are necessary, and they are best possible. 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, namely the "unit in the first place (ufp)", which cures drawbacks of the often-used "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. Goal of the paper. Let $\mathbb{F}$ denote a set of binary floating-point numbers according to the IEEE 754 floating-point standard [4, 5]. Throughout the paper we assume that no overflow occurs, but allow underflow. Denote by float$(\cdot)$ the evaluation of an expression in floating-point arithmetic.

The aim of the paper is as follows. The error of floating-point summation or dot product is usually estimated by the sum and product of absolute values, respectively. For example, for $p \in \mathbb{F}^n$ typically [3]

$$\Delta := |\mathrm{float}(\sum p_i) - \sum p_i| \le \gamma_{n-1} \sum |p_i| \qquad \text{provided } \ n\mathbf{u} < 1 \tag{1.1}$$

is used, where $\gamma_k := k\mathbf{u}/(1-k\mathbf{u})$ for the relative rounding error unit $\mathbf{u}$. So the error of the approximation $\tilde{s} := \mathrm{float}(\sum p_i) \in \mathbb{F}$ of the true sum $\sum p_i \in \mathbb{R}$ is estimated using $E := \sum |p_i|$. This standard Wilkinson-type estimate depends on the unknown quantity E, whereas in practice only $\tilde{E} := \mathrm{float}(\sum |p_i|)$ is available. It is well-known how to close this gap (see (3.3)), however, then the formula becomes even more unwieldy.

It seems desirable to improve this in several ways, namely to develop an estimate using the known quantity $\tilde{E}$ with a straight dependence on n and $\mathbf{u}$, to lift the restriction on n, the estimate should be computable and, if possible, should be sharp. We will show

$$\Delta \le (n-1)\mathbf{u}\tilde{E} \quad \text{and even} \quad \Delta \le (n-1)\mathbf{u} \cdot \mathrm{ufp}(\tilde{E}) \ , \tag{1.2}$$

where $\mathrm{ufp}(\tilde{E})$, the "unit in the first place", denotes the largest power of 2 not larger than $\tilde{E}$. We show that the estimates are sharp, i.e. for the given information $\tilde{s}$ and $\tilde{E}$ they are not improvable, and (1.2) is valid for all n. It is shown how to compute the unit in the first place in four floating-point operations in rounding to nearest.

Furthermore, the second estimate in (1.2) is still valid when evaluated in floating-point arithmetic if $n\mathbf{u} \le 1$ (otherwise n need not be a floating-point number). Almost similar results are presented for dot products.

*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.

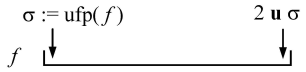

FIG. 2.1. *Normalized floating-point number: unit in the first place and unit in the last place*

2. Notation. The relative rounding error unit, the distance from 1.0 to the next smaller[1] floating-point number, is denoted by **u**, and the underflow unit by `eta`, that is the smallest positive (subnormal) floating-point number. For IEEE 754 double precision (binary64) we have $\mathbf{u} = 2^{-53}$ and $\texttt{eta} = 2^{-1074}$.

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

$$x \in \mathbb{R} \ : \ |\mathrm{fl}(x) - x| = \min\{|f - x| : f \in \mathbb{F}\} \ . \tag{2.1}$$

Any rounding of the tie can be used without jeopardizing the following estimates, only (2.1) must hold true. This implies that for rounding downwards or upwards or towards zero, all bounds are true *mutatis mutandis* using $2\mathbf{u}$ instead of **u**. This may be particularly useful for cell processors [6, 7]. For $a, b \in \mathbb{F}$ and $\circ \in \{+, -, \cdot, /\}$, the IEEE 754 floating-point standard defines $\mathrm{fl}(a \circ b) \in \mathbb{F}$ to be the floating-point approximation of $a \circ b \in \mathbb{R}$.

The standard error estimate (see, e.g., [3]) for floating-point addition and subtraction is

$$\mathrm{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2) \quad \text{for } a, b \in \mathbb{F},\ \circ \in \{+, -\} \ \text{ and } \ |\varepsilon_1|, |\varepsilon_2| \le \mathbf{u} \ . \tag{2.2}$$

Note that addition and subtraction is exact near underflow [2], so no underflow unit is necessary in (2.2). More precisely,

$$a, b \in \mathbb{F} \ : \quad |\mathrm{fl}(a \pm b)| < \mathbf{u}^{-1}\texttt{eta} \quad \Rightarrow \quad \mathrm{fl}(a \pm b) = a \pm b \ . \tag{2.3}$$

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

$$\mathrm{fl}(a \circ b) = (a \circ b)(1 + \varepsilon) + \eta \quad \text{for } a, b \in \mathbb{F},\ \circ \in \{\cdot, /\} \ \text{ and } \ |\varepsilon| \le \mathbf{u}, |\eta| \le \texttt{eta}/2, \varepsilon\eta = 0 \ . \tag{2.4}$$

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. For example,

$\mathrm{ulp}(s1) = 2^{-23}$ for $s1 = 1$ in IEEE 754 single precision (binary32),
$\mathrm{ulp}(d1) = 2^{-52}$ for $d1 = 1$ in IEEE 754 double precision (binary64),
$\mathrm{ulp}(u) = 2^{-149}$ for any $u \in \mathbb{F}$ with $2^{-149} \le |u| < 2^{-125}$ in IEEE 754 single precision (binary32),

and $\mathrm{ulp}(\pi)$ is not defined at all.

Therefore I introduced in [10] the "unit in the first place (ufp)" or leading binary bit of a real number, which is defined by

$$0 \ne r \in \mathbb{R} \quad \Rightarrow \quad \mathrm{ufp}(r) := 2^{\lfloor \log_2 |r| \rfloor} \ , \tag{2.5}$$

where $\mathrm{ufp}(0) := 0$. 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 $\mathrm{ufp}(f)$ and the unit in the last place $2\mathbf{u} \cdot \mathrm{ufp}(f)$. In particular $f \in 2\mathbf{u} \cdot \mathrm{ufp}(f)\mathbb{Z}$ and $\mathbb{F} \subset \texttt{eta}\mathbb{Z}$, also in underflow. This interpretation of floating-point numbers as scaled integers turns out to be very useful. The situation is depicted in Figure 2.1.

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

Many interesting properties of ufp(·) are given in [10] 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.

$$0 \neq x \in \mathbb{R} \;:\quad \mathrm{ufp}(x) \le |x| < 2\mathrm{ufp}(x) \tag{2.6}$$

$$x \in \mathbb{R},\ f \in \mathbb{F} \;:\quad |x - f| < \mathbf{u} \cdot \mathrm{ufp}(x) \quad \Rightarrow \quad \mathrm{fl}(x) = f \tag{2.7}$$

$$r \in \mathbb{R} \quad \Rightarrow \quad \mathrm{ufp}(r) \le \mathrm{ufp}(\mathrm{fl}(r)) \ . \tag{2.8}$$

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

$$f = x + \delta + \eta \ \text{ with } \ |\delta| \le \mathbf{u} \cdot \mathrm{ufp}(x) \le \mathbf{u} \cdot \mathrm{ufp}(f) \le \mathbf{u}|f|,\ |\eta| \le \mathtt{eta}/2,\ \delta\eta = 0 \ . \tag{2.9}$$

This implies in particular for floating-point addition and multiplication

$$f = \mathrm{fl}(a + b) \quad \Rightarrow \quad f = a + b + \delta \ \text{ with } \ |\delta| \le \mathbf{u} \cdot \mathrm{ufp}(a + b) \le \mathbf{u} \cdot \mathrm{ufp}(f) \ , \tag{2.10}$$

$$f = \mathrm{fl}(a \cdot b) \ \Rightarrow \ f = a \cdot b + \delta + \eta \ \text{ with } \ |\delta| \le \mathbf{u} \cdot \mathrm{ufp}(a \cdot b) \le \mathbf{u} \cdot \mathrm{ufp}(f),\ |\eta| \le \mathtt{eta}/2 \ , \tag{2.11}$$

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

The improvement by the ufp-concept in (2.9) over the classical bounds (2.2) and (2.4) is up to a factor 2. For example, $\delta := |\mathrm{fl}(3/100) - 3/100| \le (3/100)\mathbf{u} = 3.33 \cdot 10^{-18}$ by (2.4) compared to $\delta \le \mathrm{ufp}(3/100)\mathbf{u} = 1.73 \cdot 10^{-18}$ by (2.9) in IEEE 754 double precision.

3. 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 3.1. *Recursive summation of a vector p_i of floating-point numbers.*

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

For $n\mathbf{u} \le 1$, the computed approximation $\widetilde{s}_n$ satisfies the well-known error estimate (cf. [3], Lemma 8.4)

$$|\widetilde{s}_n - \sum_{i=1}^{n} p_i| \le \gamma_{n-1} \sum_{i=1}^{n} |p_i| \quad \text{with} \quad \gamma_k := \frac{k\mathbf{u}}{1 - k\mathbf{u}} \ , \tag{3.1}$$

which follows by recursively applying (2.2). The quantity γ_{n-1} takes care of accumulated rounding errors. Although this is a mathematically clean principle, it lacks computational simplicity.

As has been noted by one of the anonymous referees, under the stronger assumption $n\mathbf{u} \le 0.01$ the right hand side in (3.1) can be replaced by $1.01(n-1)\mathbf{u}\sum_{i=1}^{n} |p_i|$ (cf. [3], page 68). Although this avoids $\mathcal{O}(\mathbf{u}^2)$ terms, it uses the typical Wilkinson-style 1.01-factor, which was avoided by the γ_k-notation.

To receive a computable estimation, it gets even worse:

ALGORITHM 3.2. *Recursive summation with error bound estimation.*

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

The estimate (3.1) 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 (3.1) to $\sum |p_i|$ yields

$$|\widetilde{s}_n - \sum_{i=1}^{n} p_i| \leq \gamma_{n-1} S \leq \gamma_{n-1}\big[\ |\widetilde{S}_n| + |\widetilde{S}_n - S|\ \big] \leq \gamma_{n-1}\big[\ |\widetilde{S}_n| + \gamma_{n-1} S\ \big]\ , \tag{3.2}$$

so that

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

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

$$|\widetilde{s}_n - \sum_{i=1}^{n} p_i| \leq \frac{(n-1)\mathbf{u}}{1 - 2(n-1)\mathbf{u}}\ \widetilde{S}_n\ . \tag{3.3}$$

In order to arrive at a computable floating-point number bounding the error, extra care is necessary when evaluating the right hand side of (3.3).

The classical bound (3.1) can be freed from the nasty $\mathcal{O}(\mathbf{u}^2)$ term. To show this we need the following auxiliary lemma. Although the result is obvious after drawing a picture and a little thinking, note the simplicity and beauty of the proof using the ufp-concept.

LEMMA 3.3. *Let $a, b \in \mathbb{F}$. Then*

$$|\mathrm{fl}(a+b) - (a+b)| \leq |b|\ . \tag{3.4}$$

PROOF. If $|(a+b) - a| < \mathbf{u} \cdot \mathrm{ufp}(a+b)$, then (2.7) implies $\mathrm{fl}(a+b) = a$ and therefore (3.4). If $\mathrm{fl}(a+b) \neq a$, then (2.10) yields

$$|b| = |(a+b) - a| \geq \mathbf{u} \cdot \mathrm{ufp}(a+b) \geq |\mathrm{fl}(a+b) - (a+b)|\ . \qquad \square$$

THEOREM 3.4. *Let $p_i \in \mathbb{F}$ for $1 \leq i \leq n$, and let $\widetilde{s}_n$ be the quantity computed by recursive summation as in Algorithm 3.1. Then*

$$|\ \widetilde{s}_n - \sum_{i=1}^{n} p_i\ | \leq (n-1)\boldsymbol{u} \sum_{i=1}^{n} |p_i|\ . \tag{3.5}$$

REMARK 1. Note that in contrast to (3.1) there is no restriction on n, whereas the classical bound (3.1) becomes very weak for $n\mathbf{u}$ close to 1.

PROOF. Proceeding by induction we note

$$\Delta := |\ \widetilde{s}_n - \sum_{i=1}^{n} p_i\ | = |\widetilde{s}_n - s_n + \widetilde{s}_{n-1} - \sum_{i=1}^{n-1} p_i| \leq |\widetilde{s}_n - s_n| + (n-2)\mathbf{u} \sum_{i=1}^{n-1} |p_i|\ . \tag{3.6}$$

We distinguish two cases. First, assume $|p_n| \leq \mathbf{u} \sum_{i=1}^{n-1} |p_i|$. Then Lemma 3.3 implies

$$|\widetilde{s}_n - s_n| = |\mathrm{fl}(\widetilde{s}_{n-1} + p_n) - (\widetilde{s}_{n-1} + p_n)| \leq |p_n| \leq \mathbf{u} \sum_{i=1}^{n-1} |p_i|\ , \tag{3.7}$$

and inserting into (3.6) finishes this part of the proof. Henceforth, assume $\mathbf{u}\sum_{i=1}^{n-1}|p_i| < |p_n|$. Then (2.9) and (2.6) give

$$|\widetilde{s}_n - s_n| \le \mathbf{u}|s_n| = \mathbf{u}|\widetilde{s}_{n-1} - \sum_{i=1}^{n-1} p_i + \sum_{i=1}^{n} p_i| \ ,$$

so that (3.6) and the induction hypothesis yield

$$\begin{aligned} \Delta &\le \mathbf{u}\big[\ (n-2)\mathbf{u}\textstyle\sum_{i=1}^{n-1}|p_i| + \sum_{i=1}^{n}|p_i|\ \big] + (n-2)\mathbf{u}\sum_{i=1}^{n-1}|p_i| \\ &< \mathbf{u}\big[\ (n-2)|p_n| + |p_n| + \textstyle\sum_{i=1}^{n-1}|p_i|\ \big] + (n-2)\mathbf{u}\sum_{i=1}^{n-1}|p_i| \\ &= (n-1)\mathbf{u}\textstyle\sum_{i=1}^{n}|p_i| \ . \end{aligned}$$

The proof is finished. □

REMARK 2. The estimate (3.5) is almost sharp as for $p_1 = 1$ and $p_i = \mathbf{u}$ for $2 \le i \le n$. There are obvious possibilities to improve it, in particular p_n does not occur in (3.7); however, this would sacrifice its simplicity.

For a computable bound, we may proceed as in (3.2). For $\widetilde{S}$ denoting a floating-point approximation of $\sum|p_i|$ computed by Algorithm 3.1 in any order of summation, Theorem 3.4 yields

$$|\widetilde{s}_n - \sum_{i=1}^{n} p_i| \le (n-1)\mathbf{u}\cdot S \le (n-1)\mathbf{u}\big[\widetilde{S} + (n-1)\mathbf{u}\cdot S\big] \ ,$$

and therefore

$$|\widetilde{s}_n - \sum_{i=1}^{n} p_i| \le \gamma_{n-1}\widetilde{S} \ . \tag{3.8}$$

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

To arrive at an easily computable and nice bound, we analyze Algorithm 3.2 further. We will use a monotonicity property of a rounding defined by (2.1), namely

$$|x| \le X \text{ and } |y| \le Y \ \Rightarrow\ |\mathrm{fl}(x+y)| \le \mathrm{fl}(X+Y) \quad \text{for } \ x,y,X,Y \in \mathbb{R} \ , \tag{3.9}$$

so that

$$|\widetilde{s}_k| \le \widetilde{S}_k \quad \text{for } \ 1 \le k \le n \tag{3.10}$$

for the quantities computed by Algorithm 3.2.

THEOREM 3.5. *Let $p_i \in \mathbb{F}$ for $1 \le i \le n$, and let $\widetilde{s}_n$ and $\widetilde{S}_n$ be the quantities computed by recursive summation as in Algorithm 3.2. Then*

$$|\ \widetilde{s}_n - \sum_{i=1}^{n} p_i\ | \le (n-1)\boldsymbol{u}\cdot \mathrm{ufp}(\widetilde{S}_n) \qquad \big[\ \le (n-1)\boldsymbol{u}\widetilde{S}_n\ \big] \ . \tag{3.11}$$

The estimation is sharp for all n.

REMARK 3. As for the previous theorem we note that there is no restriction on n. Note that for the given data, namely the floating-point approximations $\widetilde{s}_n$ and $\widetilde{S}_n$, the estimate is best possible for all n.

PROOF. We use (2.9) and (3.10) to see

$$|\widetilde{s}_k - s_k| \le \mathbf{u}\cdot\mathrm{ufp}(\widetilde{s}_k) \le \mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_k)$$

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

$$\begin{aligned} |\ \widetilde{s}_k - \textstyle\sum_{i=1}^{k} p_i\ | &= |\ \widetilde{s}_k - s_k + \widetilde{s}_{k-1} - \sum_{i=1}^{k-1} p_i\ | \\ &\le \mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_k) + (k-2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_{k-1}) \\ &\le (k-1)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_k) \ . \end{aligned}$$

This proves (3.11). Setting $p_1 := 1$ and $p_i := \mathbf{u}$ for $2 \le i \le n$ we obtain in IEEE 754 rounding tie to even $\mathrm{fl}(1+\mathbf{u}) = 1$, so that $\widetilde{s}_n = 1 = \widetilde{S}_n$. Hence $|\ \widetilde{s}_n - \sum_{i=1}^n p_i\ | = (n-1)\mathbf{u} = (n-1)\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n)$. This completes the proof. □

The bound in (3.11), unlike (3.3) and (3.8), is only valid when both $\widetilde{s}_n$ and $\widetilde{S}_n$ are computed in the same order of summation. Concerning a practical application of (3.11), the unit in the first place can be computed with 4 floating-point operations. Algorithm 3.5 in [9], 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 3.6. *Unit in the first place of a floating-point number.*

$$\begin{array}{ll} \text{function } S = \mathrm{ufp}(p) & \\ \quad q = \mathrm{fl}(\varphi\, p) & \text{for } \varphi := (2\boldsymbol{u})^{-1} + 1 \\ \quad S = \mathrm{fl}(|q - (1-\boldsymbol{u})q|) & \end{array}$$

In combination with Theorem 3.4 this allows an easy computation of a rigorous error bound for floating-point summation, whereas a rigorous bound based on (3.3) or (3.8) in rounding to nearest is unwieldy.

Corollary 3.7. *Let $p_i \in \mathbb{F}$ for $1 \le i \le n$, assume $n\boldsymbol{u} \le 1$, and let $\widetilde{s}_n, \widetilde{S}_n$ and $r := \mathrm{ufp}(\widetilde{S}_n)$ be the quantities computed by Algorithms 3.2 and 3.6. Then*

$$|\ \widetilde{s}_n - \sum_{i=1}^n p_i\ | \le \mathrm{fl}\big(\ (n-1) \cdot \mathrm{fl}(\boldsymbol{u} \cdot r)\ \big)\ . \tag{3.12}$$

The estimation is sharp for all $n \le \boldsymbol{u}^{-1}$.

Remark 4. Note that n is not necessarily a floating-point number for $n > \mathbf{u}^{-1}$. It is straightforward (though of small practical interest) to derive a computable upper bound for arbitrary values of n.

Proof. Denote the floating-bound in (3.12) by $\widetilde{R}$. If $\widetilde{S}_n < \mathbf{u}^{-1}\mathtt{eta}$, then (3.10) and (2.3) imply that there is no rounding error at all in the summations producing $\widetilde{s}_n$ and $\widetilde{S}_n$, i.e. $\widetilde{s}_n = \sum p_i$. In rounding tie to even also $\mathrm{fl}(\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n)) = \mathrm{fl}(\mathtt{eta}/2) = 0 = \widetilde{R}$.

Otherwise, since $\mathrm{ufp}(\widetilde{S}_n)$ is a power of 2, $\mathrm{ufp}(\widetilde{S}_n) \ge \mathbf{u}^{-1}\mathtt{eta}$ implies that for n not larger than $\mathbf{u}^{-1}$ no rounding error occurs in the computation of $\widetilde{R}$, so Theorem 3.5 finishes the proof. □

Remark 5. One might be inclined to improve Theorem 3.4 along the lines of Theorem 3.5 into

$$|\ \widetilde{s}_n - \sum_{i=1}^n p_i\ | \le (n-1)\mathbf{u} \cdot \mathrm{ufp}(\sum_{i=1}^n |p_i|)\ .$$

However, this is not true as by

$$p_{1\ldots5} := [\ 1 - 5\mathbf{u}\ ,\ \frac{\mathbf{u}}{2}\ ,\ \frac{3}{2}\mathbf{u}\ ,\ \frac{3}{2}\mathbf{u}\ ,\ \mathbf{u}(1+2\mathbf{u})\ ]\ .$$

First note that $p_i \in \mathbb{F}$. Second, $\widetilde{S}_{2\ldots5} = \widetilde{s}_{2\ldots5} = 1 + \mathbf{u} \cdot [-4, -2, 0, 2]$ and $s := \sum p_i = 1 - \frac{\mathbf{u}}{2} + 2\mathbf{u}^2$. Hence

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

but

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

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

ALGORITHM 4.1. *Dot product of two vectors $x = (x_1, \ldots, x_n)^T$, $y = (y_1, \ldots, y_n)^T$ of floating-point numbers.*

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

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

$$
|\widetilde{s}_n - x^T y| \le \gamma_n |x^T||y| \ , \tag{4.1}
$$

where the absolute value of vectors is taken entrywise. As in the remark following (3.1) the right hand side may be replaced by $1.01n\mathbf{u}|x^T||y|$ under the stronger assumption $n\mathbf{u} \le 0.01$. Next we extend Algorithm 4.1 to obtain a computable error estimate including underflow.

ALGORITHM 4.2. *Dot product of two vectors $x, y \in \mathbb{F}^n$ with error bound estimate.*

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

Using the results of the previous section we derive the following computable error estimate for the floating-point approximation $\widetilde{s}_n$ computed by Algorithm 4.2 to the exact dot product $x^T y$.

THEOREM 4.3. *Let $x, y \in \mathbb{F}^n$ with $(n+2)\mathbf{u} \le 1$, and let $\widetilde{s}_n$ and $\widetilde{S}_n$ be the quantities computed by Algorithm 4.2. Denote by* $\texttt{realmin} := \frac{1}{2}\mathbf{u}^{-1}\texttt{eta}$ *the smallest positive normalized floating-point number. Then*

$$
| \widetilde{s}_n - x^T y | < (n+2)\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n) + n\texttt{eta}/2 < (n+2)\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n) + \texttt{realmin} \ . \tag{4.2}
$$

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

REMARK 6. It is not possible to avoid some additive term covering underflow in (4.2) since all $|x_i y_i|$ may be so small that $\widetilde{s}_n = \widetilde{S}_n = 0$ but $x^T y \neq 0$.

REMARK 7. Floating-point operations with quantities in the underflow range (such as $n\texttt{eta}/2$) are often very time consuming, so that for computational purposes the underflow terms are better estimated by `realmin` rather than $n\texttt{eta}/2$. Moreover, this quantity is negligible in most cases anyway.

PROOF. Applying Theorem 3.5 to the vector $(|\widetilde{p}_1|, \ldots, |\widetilde{p}_n|)^T \in \mathbb{F}^n$ and using $n\mathbf{u} \le 1$ gives

$$
| \widetilde{S}_n - \sum_{i=1}^{n} |\widetilde{p}_i| \ | \ \le \ (n-1)\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n) < \mathrm{ufp}(\widetilde{S}_n) \ , \tag{4.3}
$$

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

$$
| \widetilde{s}_n - x^T y | = | \widetilde{s}_n - \sum_{i=1}^{n} p_i \ | \ \le \ (n-1)\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n) + | \sum_{i=1}^{n} (\widetilde{p}_i - p_i) \ | \ . \tag{4.4}
$$

Next (2.9) yields

$$|\widetilde{p}_i - p_i| = |\mathrm{fl}(p_i) - p_i| \le \mathbf{u} \cdot |\widetilde{p}_i| + \mathtt{eta}/2 \ ,$$

so that (4.3) and (2.6) imply

$$\begin{array}{rcl} |\sum_{i=1}^n (\widetilde{p}_i - p_i)\,| & \le & \mathbf{u} \sum_{i=1}^n |\widetilde{p}_i| + n\mathtt{eta}/2 \\ & < & \mathbf{u}\big[\widetilde{S}_n + \mathrm{ufp}(\widetilde{S}_n)\big] + n\mathtt{eta}/2 \\ & < & 3\mathbf{u} \cdot \mathrm{ufp}(\widetilde{S}_n) + n\mathtt{eta}/2 \ . \end{array}$$

With (4.4) and $n < \mathbf{u}^{-1}$ this proves (4.2). 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. $\mathbf{u} = 2^{-k}$, define

$$x_i := \begin{cases} 2^{-1}(1+2\mathbf{u}) & \text{for } \ i = 1 \\ 2^{-i}(1+2\mathbf{u}) + \frac{\mathbf{u}}{2} & \text{for } \ 2 \le i \le k \\ \frac{\mathbf{u}}{2}(1+2\mathbf{u}) & \text{for } \ k+1 \le i \le n \end{cases} \tag{4.5}$$

and $y_{1\ldots n} := 1 - \mathbf{u}$. Note that $x_i, y_i \in \mathbb{F}$. Applying Algorithm 4.2 yields

$$\widetilde{p}_1 = 2^{-1} \ , \ \ \widetilde{p}_i := 2^{-i} + \frac{\mathbf{u}}{2} \ \text{ for } 2 \le i \le k \ \ \text{ and } \ \widetilde{p}_i := \frac{\mathbf{u}}{2} \ \text{ for } k+1 \le i \le n \ .$$

Therefore

$$\widetilde{S}_n = \widetilde{s}_n = \sum_{i=1}^k 2^{-i} = 1 - \mathbf{u} \quad \text{and} \quad \ \mathrm{ufp}(\widetilde{S}_n) = \frac{1}{2} \ .$$

But

$$\begin{array}{rcl} x^T y & = & (1-\mathbf{u}) \sum_{i=1}^n x_i \\ & = & (1-\mathbf{u})\big[\ (1-\mathbf{u})(1+2\mathbf{u}) + (k-1)\frac{\mathbf{u}}{2} + (n-k)\frac{\mathbf{u}}{2}(1+2\mathbf{u})\ \big] \\ & = & (1-\mathbf{u})\big[\ 1 + (n+1)\frac{\mathbf{u}}{2} + (n-k-2)\mathbf{u}^2\ \big] \\ & > & 1 + (n-1)\frac{\mathbf{u}}{2} + (n-2k-5)\frac{\mathbf{u}^2}{2} \ , \end{array}$$

so that

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

Therefore, in view of $\mathrm{ufp}(\widetilde{S}_n) = \frac{1}{2}$, the factor $n+2$ in (4.2) 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. □

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

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

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 4.4. *Let $x, y \in \mathbb{F}^n$ with $2(n+2)\boldsymbol{u} \le 1$ be given, denote by $\mathtt{realmin} := \frac{1}{2}\boldsymbol{u}^{-1}\mathtt{eta}$ the smallest positive normalized floating-point number, and let $\widetilde{s}_n, \widetilde{S}_n$ and $r := \mathrm{ufp}(\widetilde{S}_n)$ be the quantities computed by Algorithm 4.2. Define*

$$\widetilde{R} := \mathit{float}\big(\ (n+2)\cdot(\boldsymbol{u}\cdot r) + \mathtt{realmin}\ \big) \quad \textit{and} \quad \hat{R} := \mathit{float}\big(\ ((n+2)\cdot\boldsymbol{u})\cdot r + \mathtt{realmin}\ \big) \ , \tag{4.6}$$

where float(·) means that all operations within the parenthesis are carried out in floating-point arithmetic rounded to nearest in the specified order. Then

$$|\,\widetilde{s}_n - x^T y|\ \le\ \widetilde{R} \quad \textit{and} \quad |\,\widetilde{s}_n - x^T y|\ \le\ \hat{R}\ . \tag{4.7}$$

If only $(n+2)\mathbf{u} \le 1$ *is satisfied, then (4.7) remains valid when replacing* `realmin` *in the computation of* $\widetilde{R}$ *and* $\hat{R}$ *by* $\varrho := \frac{3}{2}$`realmin`*, respectively.*

Remark 8. Note that $\mathrm{fl}(n+2) = n+2$ and $\mathrm{fl}(\frac{3}{2}\cdot$`realmin`$) = \varrho$. The assumption $2(n+2)\mathbf{u} \le 1$ implies in IEEE 754 the practical limitation $n < 4.5 \cdot 10^{15}$.

Remark 9. The first quantity $\widetilde{R}$ in (4.7) is advantageous for $\mathrm{ufp}(\widetilde{S}_n)$ near the underflow range; in the underflow range $\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$ even becomes zero. When applying Corollary 4.4 to matrix multiplication, however, $\mathrm{ufp}(\widetilde{S}_n)$ 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, where in the first case ϱ can be omitted in the computation of $\tilde{R}$ and $\hat{R}$ without jeopardizing the assertions, and in the second case $\varrho :=$ `realmin` can be used even for the weaker assumption $(n+2)\mathbf{u} \le 1$. Only in the third case we have to distinguish $\varrho :=$ `realmin` and $\varrho := \frac{3}{2}$`realmin`.

First, assume $\widetilde{S}_n \ge \mathbf{u}^{-1}$`realmin`, so that $\mathrm{ufp}(\widetilde{S}_n) \ge \mathbf{u}^{-1}$`realmin`. Then both products in the floating-point computation of $(n+2)\cdot\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$ do not cause a rounding error. Therefore

$$\widetilde{R} = \hat{R} \ge (n+2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$$

because $\mathrm{fl}(a+b) \ge a$ for any nonnegative $a, b \in \mathbb{F}$. Hence, in view of (4.4), it suffices to prove

$$|\sum_{i=1}^{n}(\widetilde{p}_i - p_i)\ | \le 3\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n) \tag{4.8}$$

under the assumption $\mathrm{ufp}(\widetilde{S}_n) \ge \mathbf{u}^{-1}$`realmin`. Denote by $I \subseteq \{1,\dots,n\}$ the index set for which $|\widetilde{p}_i| <$ `realmin`. Then $\varepsilon\eta = 0$ in (2.9) yields

$$|\widetilde{p}_i - p_i|\ \le\ \begin{cases} \texttt{eta}/2 & \text{for } i \in I \\ \mathbf{u}|\widetilde{p}_i| & \text{otherwise}\ , \end{cases}$$

so that

$$|\sum_{i=1}^{n}(\widetilde{p}_i - p_i)\ | \le \mathbf{u}\sum_{i\notin I}|\widetilde{p}_i| + k\texttt{eta}/2 \qquad \text{for } k := |I|\ . \tag{4.9}$$

Denote by $\widetilde{S}_n^*$ the result of Algorithm 3.2 when applied to the vector $\{\ |\widetilde{p}_i|\ : i \notin I\ \}$, that is leaving out the vector elements in the index set I. Recursively applying (3.9) implies $0 \le \widetilde{S}_n^* \le \widetilde{S}_n$ because the order of summation is not changed. Thus Theorem 3.5, (2.6) and $n\mathbf{u} < 1$ give

$$\sum_{i\notin I}|\widetilde{p}_i|\ \le \widetilde{S}_n^* + (n-k-1)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n^*) < \big(2 + (n-k-1)\mathbf{u}\big)\cdot\mathrm{ufp}(\widetilde{S}_n) < (3-k\mathbf{u})\mathrm{ufp}(\widetilde{S}_n)\ .$$

Hence (4.9) and $\mathrm{ufp}(\widetilde{S}_n) \ge \mathbf{u}^{-1}$`realmin` yield

$$|\sum_{i=1}^{n}(\widetilde{p}_i - p_i)\ | \le \big(3-k\mathbf{u}\big)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n) + k\texttt{eta}/2 \le 3\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$$

and prove (4.8) as desired. This finishes the first case.

Second, assume $\widetilde{S}_n < \mathbf{u}^{-1}$`eta`. Then by (2.3), as already discussed, there is no rounding error at all in the summations producing $\widetilde{s}_n$ and $\widetilde{S}_n$. Then, because the result of the floating-point computation of

$(n+2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$ in any order is nonnegative, the total error $|\widetilde{s}_n - x^T y|$ is bounded by $n\mathtt{eta}/2 < \mathtt{realmin} \le \min(\widetilde{R},\hat{R})$.

It remains the third case $\mathbf{u}^{-1}\mathtt{eta} \le \widetilde{S}_n < \mathbf{u}^{-1}\mathtt{realmin}$. Then there is no rounding error in the floating-point computation of $\Phi := (n+2)\cdot\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n)$, no matter of the order of the product. As the first subcase assume $(n+2)\mathbf{u} \le 1$ and $\varrho := \frac{3}{2}\mathtt{realmin}$. Then $\Phi = \widetilde{R} = \hat{R} \le \mathrm{ufp}(\widetilde{S}_n) \le \frac{1}{2}\mathbf{u}^{-1}\mathtt{realmin}$ and

$$\mathrm{ufp}(\Phi+\varrho) \le \mathrm{ufp}\big(\ \frac{1}{2}\mathbf{u}^{-1}\mathtt{realmin} + \varrho\ \big) = \frac{1}{2}\mathbf{u}^{-1}\mathtt{realmin}\ .$$

Now using (2.10) shows

$$\widetilde{R} = \hat{R} = \mathrm{fl}(\Phi+\varrho) \ge \Phi + \varrho - \mathbf{u}\cdot\mathrm{ufp}(\Phi+\varrho) \ge (n+2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n) + \mathtt{realmin}$$

and Theorem 4.3 applies. This leaves us with the second and last subcase $2(n+2)\mathbf{u} \le 1$ and $\varrho := \mathtt{realmin}$. Then $\Phi = \widetilde{R} = \hat{R} \le \frac{1}{2}\mathrm{ufp}(\widetilde{S}_n) \le \frac{1}{4}\mathbf{u}^{-1}\mathtt{realmin}$ and

$$\mathrm{ufp}(\Phi+\varrho) \le \mathrm{ufp}\big(\ \frac{1}{4}\mathbf{u}^{-1}\mathtt{realmin} + \varrho\ \big) = \frac{1}{4}\mathbf{u}^{-1}\mathtt{realmin}\ .$$

Again using (2.10) and $\mathtt{realmin} = \frac{1}{2}\mathbf{u}^{-1}\mathtt{eta} > n\mathtt{eta}$ shows

$$\begin{aligned}\widetilde{R} = \hat{R} &= \mathrm{fl}(\Phi+\varrho) \ge \Phi + \varrho - \mathbf{u}\cdot\mathrm{ufp}(\Phi+\varrho)\\ &\ge (n+2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n) + \tfrac{3}{4}\mathtt{realmin}\\ &> (n+2)\mathbf{u}\cdot\mathrm{ufp}(\widetilde{S}_n) + \tfrac{1}{2}n\mathtt{eta}\ ,\end{aligned}$$

and Theorem 4.3 applies again. The proof is finished. □

We note again that the way Corollary 4.4 handles the underflow uses the computational advantage that a *normalized* number is added to the computed result $\mathrm{ufp}(\widetilde{S}_n)$: floating-point arithmetic in the underflow range on today's architectures is sometimes 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 [8] propose to compute the rowwise and comlumnwise 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 $\mathcal{O}(n^2)$.

5. Practical improvement over the classical Wilkinson-type bound. Following we first compare the standard Wilkinson-type estimate (3.3) with the new computable bound (3.12) in Corollory 3.7. For simplicity we compute (3.3) 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 (3.12) 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 $\mathrm{ufp}(\widetilde{S}_n) = 4096$, so that the ratio between (3.3) and (3.12) will probably be almost 2.

To arrive at sufficiently random input data, we use vectors `p=randn(1,K)*(rand(K)*randn(K,n))` in Matlab notation[2] 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 [11]. For 1000 tests each, the

[2]The Matlab function `rand` produces pseudo-random values drawn from a uniform distribution on the unit interval, whereas `randn` produces pseudo-random values drawn from a normal distribution with mean zero and standard deviation one.

TABLE 5.1
Minimum, mean, median and maximum ratio between the estimates (3.3) with (3.12).

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 5.2
Minimum, mean, median and maximum ratio between the estimates (4.1) with (4.6).

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

minimum, mean, median and maximum ratio between the estimates (3.3) with (3.12) is displayed in Table 5.1.

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 (4.1) with the new computable one (4.6) in Corollary 4.4 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 prescribed condition number and 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 5.2, where the Wilkinson-type bound (4.1) is computed in floating-point ignoring rounding errors. We did not observe a significant difference for varying condition numbers, so it is 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 (4.6) compared to $\gamma_n \approx n$ in (4.1) the new estimate for dot products may also be worse than the traditional one. This might be fixed by taking the minimum.

The data suggests that the improvement of the presented estimates are limited to some 40% improvement of certain error bounds. However, in addition to the appealing simplicity of the bounds there are situations where the improvement changes failure into success. We finish the paper with two such examples, one for dense and one for sparse matrices.

The non-singularity of a given matrix $A \in \mathbb{F}^{n\times n}$ can be proved by computing an approximate inverse R (e.g. by the Matlab-command `R=inv(A)`) and checking $\|I - RA\| < 1$ for some norm $\|\cdot\|$, where I denote the identity matrix. Often the spectral norm is used and, because it is to time consuming to compute it directly, it is estimated by $\|C\|_2 \le \sqrt{\|C\|_1\|C\|_\infty}$. This implies the sufficient criterion

$$C := |I - RA| \quad \text{and} \quad \sqrt{\|C\|_1\|C\|_\infty} < 1 \quad \Rightarrow \qquad \det(A) \neq 0\ . \tag{5.1}$$

The application on a digital computer requires in particular an entrywise rigorous upper bound of $|I - RA|$, and of its 1- and ∞-norm. The first problem is solved by Corollary 4.4, and the second by Corollary 3.7. Alternatively, we may use the Wilkinson-type estimates (4.1) and (3.3). Extra care is necessary for both

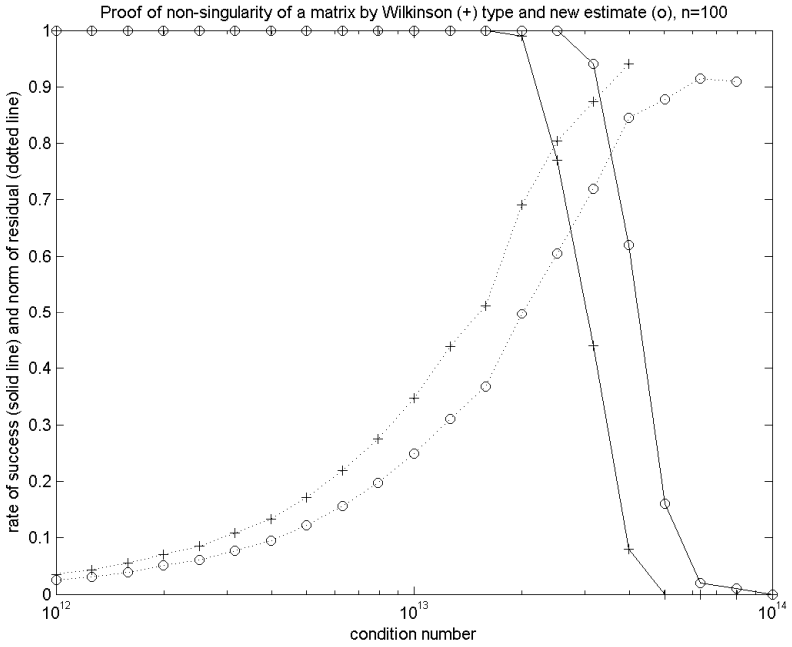

FIG. 5.1. *Percentage that* $\|I - RA\| < 1$ *is satisfied using Wilkinson-type estimates (3.3), (4.1) depicted by (+), and new estimates (4.6), (3.3) depicted by (o), dimension* $n = 100$.

to cover rounding errors when evaluating the expressions, and the first one needs to be rewritten along the lines of (3.2) and (3.3) so that the approximation $\mathrm{float}(|x|^T|y|) \in \mathbb{F}$ can be used rather than the exact value $|x|^T|y| \in \mathbb{R}$.

The two approaches are tested using the Matlab routine `randsvd` to generate random matrices with prescribed condition number via the singular value decomposition: For an anticipated condition number κ, a vector v of length n of logarithmically equidistantly distributed numbers between 1 and $1/\kappa$ is generated, and the diagonal matrix with v on the diagonal is multiplied from the left and right by a random orthogonal matrix. The order of the elements is not far from 1 in absolute value, so that we can safely ignore underflow for the Wilkinson-type estimates.

For condition number 10^k we can expect about k correct digits in the approximate inverse R, and the norm of the residual $\|I - RA\|$ can be expected to be of the order $n^{3/2}c \cdot \mathbf{u}$. Thus for condition numbers roughly up to $\mathbf{u}^{-1}/n^{3/2}$ the proof of non-singularity should be successful.

In Figure 5.1 the results for dimension $n = 100$ are displayed, where "+" is used for the Wilkinson-type bounds and "o" for the new bounds. In IEEE 754 double precision the method should be successful up to condition number 10^{13}. For each condition number we used 100 test cases. The solid line depicts the percentage of success, i.e. the input matrix was proved to be non-singular, and the dotted line depicts the median of the upper bounds of $\sqrt{\|C\|_1\|C\|_\infty}$.

As can be seen for a condition number around $3 \cdot 10^{13}$ the behavior of the methods changes from success to failure. The interesting detailed data in that area is displayed in Table 5.3.

In Figure 5.2 similar results for dimension $n = 1000$ are displayed. In IEEE 754 double precision the method should be successful up to condition number $3 \cdot 10^{11}$. The detailed data is shown in Table 5.4. For condition number $6.3 \cdot 10^{11}$, for example, the new estimate was successful in all cases, whereas the Wilkinson-type bound was successful in 7% of the 100 test cases.

The second example concerns the proof of positive definiteness. Let a symmetric matrix A be given, and assume that the floating-point Cholesky decomposition runs to completion producing some matrix G. Al-

TABLE 5.3
Wilkinson-type versus new estimates as in Figure 5.1: detailed results, dimension $n = 100$.

	median(norm bound)		percentage of success	
cond(A)	Wilkinson (3.3), (4.1)	new (4.6), (3.3)	Wilkinson (3.3), (4.1)	new (4.6), (3.3)
$2.0 \cdot 10^{13}$	0.69	0.49	99	100
$2.5 \cdot 10^{13}$	0.80	0.60	77	100
$3.2 \cdot 10^{13}$	0.87	0.72	44	94
$4.0 \cdot 10^{13}$	0.94	0.84	8	62

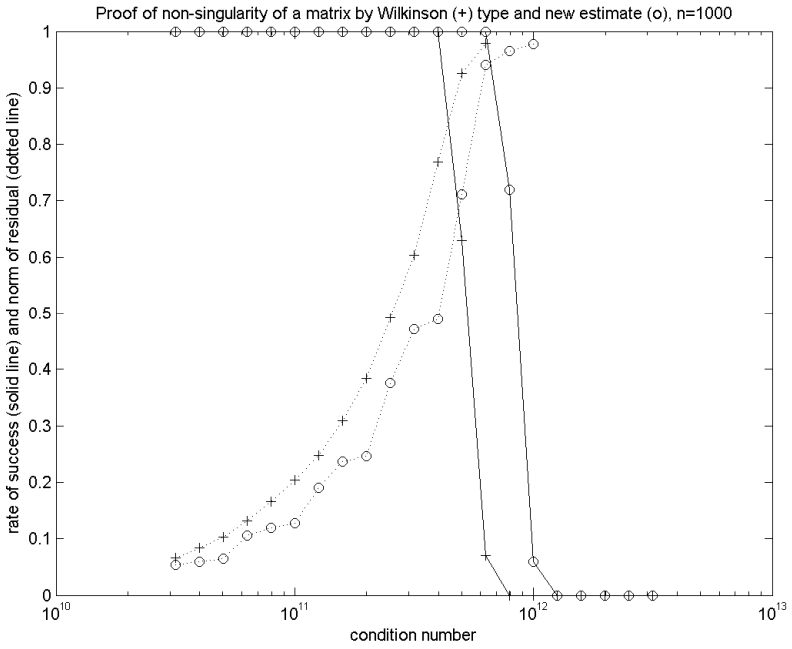

FIG. 5.2. *Percentage that $\|I - RA\| < 1$ is satisfied using Wilkinson-type estimates (3.3), (4.1) depicted by (+), and new estimates (4.6), (3.3) depicted by (o), dimension $n = 1000$.*

though it is well-known that the Cholesky decomposition is a numerically very stable algorithm, and moreover explicit bounds for the residual $\|A - G^T G\|_2$ only depending on the diagonal of A are known [1], more care is necessary to prove A to be positive definite.

Given $A^T = A \in \mathbb{F}^{n \times n}$, a simple way is to pick some $0 < \alpha \in \mathbb{R}$, and to compute an approximate Cholesky decomposition G of $A - \alpha I$. If $\|A - \alpha I - G^T G\|_2 < \alpha$, then

$$\lambda_{\min}(A) = \lambda_{\min}(A - \alpha I) + \alpha > \lambda_{\min}(A - \alpha I) + \|A - \alpha I - G^T G\|_2 \geq \lambda_{\min}(G^T G) \geq 0 \tag{5.2}$$

using Wilkinson's perturbation bound for eigenvalues of symmetric matrices, so that A is proved to be symmetric positive definite.

We take a discrete Laplacian operator on a two-dimensional $n \times n$ grid resulting in an $n^2 \times n^2$ sparse matrix. Although at most 5 entries per row in A are nonzero, the Cholesky factor is a banded matrix of bandwidth n. Based on an approximation of the smallest singular value of A, a bisection scheme is used to determine some α with $\|A - \alpha I - G^T G\|_2 < \alpha$.

Table 5.5 shows the computational results in IEEE 754 single precision. This is only an example to show the difference between the traditional and new estimates; for solving a linear system with the Poisson matrix, of course, much better methods are available. Treating larger sparse matrices in single precision becomes popular due to memory restrictions. In the application of the traditional and new bounds we used the fact

TABLE 5.4
Wilkinson-type versus new estimates as in Figure 5.2: detailed results, dimension $n = 1000$.

	median(norm bound)		percentage of success	
cond(A)	Wilkinson (3.3), (4.1)	new (4.6), (3.3)	Wilkinson (3.3), (4.1)	new (4.6), (3.3)
$5.0 \cdot 10^{11}$	0.93	0.71	63	100
$6.3 \cdot 10^{11}$	0.98	0.94	7	100
$7.9 \cdot 10^{11}$	—	0.96	0	72

TABLE 5.5
Wilkinson-type [(3.3), (4.1)] *versus new* [(4.6), (3.3)] *estimates to prove positive definiteness of the discrete Laplacian operator in IEEE 754 single precision.*

			upper bound of $\|A - \alpha I - G^T G\|_2$		proof of positive definiteness	
n	dim(A)	α	Wilkinson	new	Wilkinson	new
230	52900	$2.96 \cdot 10^{-4}$	$2.92 \cdot 10^{-4}$	$2.09 \cdot 10^{-4}$	✓	✓
240	57600	$2.72 \cdot 10^{-4}$	$3.04 \cdot 10^{-4}$	$2.18 \cdot 10^{-4}$	-	✓
250	62500	$3.09 \cdot 10^{-4}$	$3.43 \cdot 10^{-4}$	$2.46 \cdot 10^{-4}$	-	✓
260	67600	$2.86 \cdot 10^{-4}$	$3.55 \cdot 10^{-4}$	$2.55 \cdot 10^{-4}$	-	✓
270	72900	$2.65 \cdot 10^{-4}$	$3.68 \cdot 10^{-4}$	$2.64 \cdot 10^{-4}$	-	✓

that each dot product in $A - \alpha I - G^T G$ consists of at most n nonzero elements.

As can be seen, the verification of positive definiteness succeeds with the traditional Wilkinson-type bounds up to matrix dimension 52900, whereas with the new bounds up to matrix dimension 72900. We mention that a computation using directed rounding, which is mandatory in the IEEE 754 standard [4, 5], yields significantly better results. The change of the rounding mode, however, is on some architectures involved and makes computational code less portable.

6. Conclusion. New bounds are derived for the floating-point approximation of a sum and dot product. They 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. However, the new bounds are to be computed in the same order as the summation or dot product, whereas the classical bounds allow any order of computation.

As is well-known the worst case is very rare, so that both types of bounds usually grossly overestimate the true error. Nevertheless, given the available information, the new bounds are optimal, they 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 were 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.

Acknowledgement. My thanks to Florian Bünger, Stef Graillat, Claude-Pierre Jeannerod and Takeshi Ogita for helpful comments. Moreover I am grateful to the remarks of the two anonymous referees.

REFERENCES

[1] J.B. Demmel. On floating point errors in Cholesky. LAPACK Working Note 14 CS-89-87, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1989.

[2] J.R. Hauser. Handling floating-point exceptions in numeric programs. *ACM Trans. Program. Lang. Syst.*, 18(2):139–174, 1996.

[3] N. J. Higham. *Accuracy and stability of numerical algorithms.* SIAM Publications, Philadelphia, 2nd edition, 2002.

[4] *ANSI/IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic*. New York, 1985.

[5] *ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic*. New York, 2008.

[6] 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.

[7] 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.

[8] T. Ogita and S. Oishi. Fast Inclution of Interval Matrix Multiplication. *Reliable Computing*, 11:191–205, 2005.

[9] S.M. Rump. Ultimately Fast Accurate Summation. *SIAM Journal on Scietific Computing (SISC)*, 31(5):3466–3502, 2009.

[10] 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.

[11] L.N. Trefethen and R. Schreiber. Average-case stability of gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(3):335–360, 1990.