International workshop on verified computations
and related topics
University of Karlsruhe, Germany, 7-10 March, 2009

Error-Free Transformations and ill-conditioned problems

Siegfried M. Rump

Hamburg University of Technology and Waseda University, Tokyo

(joint work with Takeshi Ogita, Tokyo Woman's Christian University and Shin'ichi Oishi, Waseda University, Tokyo)

The concept of error-free transformations is rather old, but gained much attention in recent years. Quite a number of algorithms, most prominently algorithms to compute the sum or dot product of vectors of floating-point numbers, were developed recently using error-free transformations.

The appeal is that problems such as the sum of floating-point numbers can be transformed into other, simpler problems without any error. Successive transformations simplify the problem more and more until eventually it can be solved with maximum accuracy.

Following we describe some of those algorithms, in particular for the sum of floating-point numbers, one of the most basic and important problems in numerical computations. We mention that those algorithms can be used to compute the inverse of an arbitrarily ill-conditioned matrix in pure double precision floating-point arithmetic [8, 10].

One of the earliest error-free transformations [4] is the transformation of the sum $a+b$ of two floating-point numbers a, b into the sum $x+y$ of two floating-point numbers. Here x is the rounded-to-nearest floating-point sum $\mathrm{fl}(a+b)$ and $y = a+b-\mathrm{fl}(a+b)$ is the exact error. The following algorithm due to Knuth [4] performs this transformation. Note that only ordinary floating-point addition and subtraction is used to compute the exact error y.

Algorithm 1. *Error-free transformation of the sum of two floating-point numbers.*

$$\begin{array}{l}\text{function } [x, y] = \texttt{TwoSum}(a, b) \\ \quad x = \mathrm{fl}(a + b) \\ \quad z = \mathrm{fl}(x - a) \\ \quad y = \mathrm{fl}((a - (x - z)) + (b - z))\end{array}$$

The following theorem holds.

Theorem 1. *For all floating-point numbers a, b we have*

$$x = fl(a + b) \quad \textit{and} \quad x + y = a + b.$$

To prove this, two facts are important: First, the error y is always a floating-point number, and second it is computed by Algorithm `TwoSum` under all circumstances without error.

There are similar algorithms [1] for the dot product of two vectors which are based on the error-free transformation of the product $a \cdot b$ into $x = fl(a \cdot b)$ and $x + y = a \cdot b$. They use the remarkable property due to Veltkamp [1] that a 53-bit double precision

floating-point number can be split error-free into the sum of two 26-bit numbers. The trick is that in addition to the mantissa bits the sign-bit is used as well. Details can be found in [7], [12] and [13].

For the moment it suffices to know that the dot product $v^T w$ of two vectors $v, w \in \mathbb{F}^n$ can be transformed into the sum $\sum s_i$ of a vector $s \in \mathbb{F}^{2n}$ of double length. Because the transformation is error-free, it suffices to consider the summation problem.

The error-free transformation `TwoSum` can be propagated one-by-one. Given, for example, $p \in \mathbb{F}^4$, we transform

$$\begin{aligned} q_2 + \pi_2 &= p_1 + p_2 \\ q_3 + \pi_3 &= \pi_2 + p_3 \\ q_4 + \pi_4 &= \pi_3 + p_4 \end{aligned}$$

Obviously

$$\sum_{i=1}^{4} p_i = q_2 + \pi_2 + p_3 + p_4 = q_2 + q_3 + \pi_3 + p_4 = q_2 + q_3 + q_4 + \pi_4 = \sum q_i + \pi_n \ .$$

More generally, given $p \in \mathbb{F}^n$ we define $\pi_1 := p_1$ and transform

$$q_i + \pi_i = \pi_{i-1} + p_i.$$

Then

$$\sum_{i=1}^{n} p_i = \pi_n + \sum_{i=2}^{n} q_i. \tag{1}$$

The process becomes more clear in Figure 1. Here each box represents the error-free

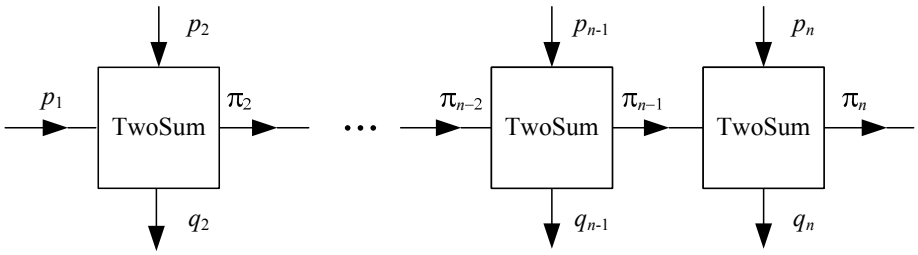

FIGURE 1. Cascade error-free Transformation `TwoSum`

transformation `TwoSum` as in Algorithm 1. Now the vector $p_1, ..., p_n$ is transformed into the vector $q_2, ..., q_n, \pi_n$, where the last element π_n is the ordinary floating-point summation $\mathrm{fl}(\sum p_i)$ and the value of the sum does not change as by (1): The transformation is error-free. Another way to write this transformation is the following [7]:

Algorithm 2. *Cascaded summation.*

function $p' =$ `VecSum`(p)
  $p'_1 = p_1$
  for $i = 2 : n$
    $[p'_i, p'_{i-1}] =$ `TwoSum`(p_i, p'_{i-1})
  end for

Here the vector $p \in \mathbb{F}^n$ is transformed into the vector $p' \in \mathbb{F}^n$ with $\sum p_i = \sum p'_i$. But not only is the transformation error-free, but the condition number $\mathrm{cond}(\sum p_i)$ decreases by almost a factor eps (the relative rounding error unit). More precisely [7],

$$\mathrm{cond}(\sum p'_i) \leq \gamma_n \cdot \mathrm{eps} \cdot \mathrm{cond}(\sum p_i),$$

where $\gamma_n := n\mathrm{eps}/(1 - n\mathrm{eps})$ [3].

This transformation process can be cascaded, each time reducing the condition number of the sum by almost a factor eps. In the last step the resulting vector is added in floating-point, the only operation which is not error-free in the entire algorithm. This algorithm `SumK` was presented in [7]:

Algorithm 3. *Summation as in K-fold precision.*

function `res` = `SumK`(p, K)
  for $k = 1 : K - 1$
    $p =$ `VecSum`(p)
  end for
  `res` $= \mathrm{fl}\left(\left(\sum_{i=1}^{n-1} p_i\right) + p_n\right)$

It can be shown [7] that the result `res` of algorithm `SumK` is essentially the same "as if" the sum would have been calculated in K-fold precision. In other words, arbitrarily ill-conditioned sums can be computed accurately to the last bit in ordinary double precision floating-point.

A drawback of Algorithm `SumK` is that the number of transformations depends on the condition number and this is not known a priori. The following algorithm `AccSum` [12], [13] calculates the sum of a vector accurate to the last bit independent of the condition number.

The idea is as follows. Figure 2 depicts the summands of a vector p_i according to

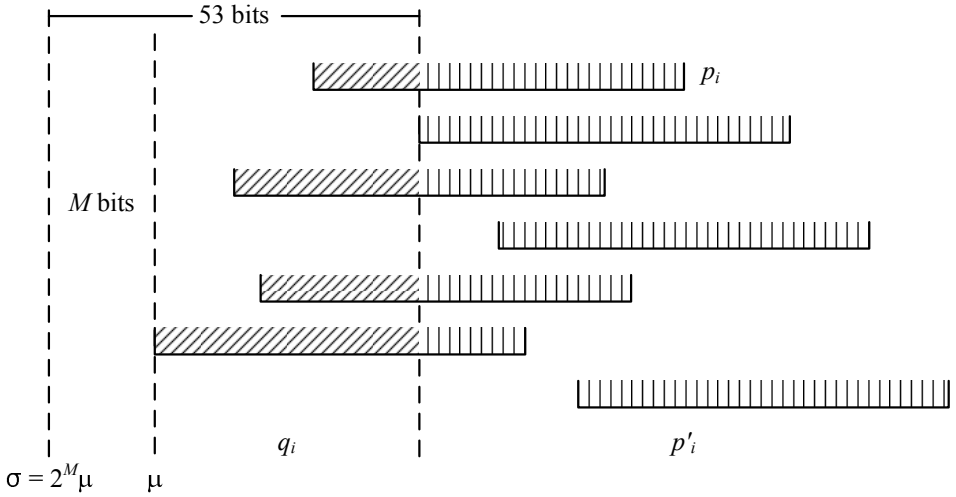

FIGURE 2. Extraction of high- and low-order parts of p_i

the bit representation of the individual elements. Denote by μ the smallest power of

2 such that $|p_i| \leq \mu$ for all i, and denote by 2^M the smallest power of 2 such that $n < 2^M$. Furthermore assume that we can extract from each p_i the bits starting with the leading bit until the bit $2^M\mu \cdot \mathrm{eps}$ as shown in Figure 1. Denote this leading part by q_i and the remaining part by p'_i, so that $p_i = q_i + p'_i$ for all i. Note that the leading part q_i may vanish, in which case $p_i = p'_i$. This happens if p_i is smaller than $2^M\mu \cdot \mathrm{eps}$. It is mandatory that the transformation $p_i = q_i + p'_i$ is error-free.

The constants are chosen in such a way that the q_i add in floating-point without error. This is true because potential carries in the sum of the q_i cannot exceed $\sigma := 2^M\mu$ because $|p_i| \leq \mu$ and $n < 2^M$. In other words,

$$\tau := \mathrm{fl}(\sum q_i) = \sum q_i,$$

the summation is error-free. Obviously this implies

$$\sum p_i = \tau + \sum p'_i. \tag{2}$$

We have to discuss how to extract the p_i into q_i and p'_i. As we will see, this can be done in only three ordinary floating-point operations.

The quantity τ can be used to decide whether the summation is well-conditioned or ill-conditioned. If τ is large, then the sum must be well-conditioned because the remaining part $\sum p'_i$ has little influence as the correction of τ. It can be shown that adding the p'_i in pure floating-point suffices to produce an approximation of the true sum being accurate to the last bit.

If, on the other hand, τ is small, then the correcting term, the sum of the p'_i may have a significant influence on τ. In fact, the final sum may even be zero. In this case we continue the process by extracting the high-order parts of the p'_i into q'_i and remaining parts p''_i. The extraction is performed in such a way that the q'_i add again in pure floating-point without error. Since $p'_i = q'_i + p''_i$ for all i, we have

$$\sum p_i = \tau + \sum p'_i = \tau + \sum q'_i + \sum p''_i.$$

Now the extraction is performed in such way that even $\tau' := \tau + \sum q'_i$ adds without error, so that finally

$$\sum p_i = \tau' + \sum p''_i. \tag{3}$$

Again we can decide on the size of τ' whether the sum of the p'_i is well-conditioned or ill-conditioned. In the first case we add $\tau' + \sum p''_i$ in pure floating-point and it can be shown that the result is accurate to the last bit. Otherwise we continue with the extraction.

An appealing property of this approach is that the computing time is proportional to the difficulty of the problem. A well-conditioned sum requires only one or few extractions, an ill-conditioned sum requires more extractions. The degree of difficulty of the problem is determined by the algorithm itself.

It remains the problem to extract the p_i into the leading, high-order part q_i and the remaining, low-order part p'_i. This extraction can be performed with the following, surprisingly simple algorithm. Given $\sigma \in \mathbb{F}$, consider

Algorithm 4. *Vector extraction.*

$$\begin{array}{l}\tau = 0 \\ \textit{for } i = 1 : n \\ \quad q_i = \mathrm{fl}((\sigma + p_i) - \sigma) \\ \quad p'_i = p_i - q_i \\ \quad \tau = \tau + q_i \\ \textit{end for}\end{array}$$

It can be shown that for every floating-point number σ satisfying $\sigma \geq |p_i|$ we have

$$p_i = q_i + p'_i, \tag{4}$$

and q_i and p'_i satisfy inequalities so that q_i is indeed the high-order part of p_i and p'_i is the low-order part. Furthermore, the subtraction $p'_i = p_i - q_i$ and the addition $\tau = \tau + q_i$ are exact, it can be shown that they do not cause a rounding error. Note again that the leading bits of p_i need not to coincide with the leading bits of q_i; the only important property is (4).

Quite some work is necessary to define σ and other constants so that the dichotomy between well-conditioned or ill-conditioned as described above is really true and, most important, that the error-free relations (2) and (3) and so forth are really satisfied. This is shown in [12]. Moreover, it is shown in [12] that the chosen constants are optimal, they cannot be improved without jeopardizing the properties of the algorithm.

In total we have a very efficient algorithm to compute the sum of floating-point numbers accurately to the last bit. Counting the number of operations of the inner loop (the vector extraction) we have 4n floating-point operations per loop. This is algorithm `AccSum` as described in [12] and [13].

There seems not much room to improve this. However, it is possible. The vector extraction above uses the same constant σ for all p_i. This is not necessary; the only necessary assumption for the mathematical properties to hold is $\sigma \geq |p_i|$ for all i. The idea in [11] is to start with some σ_0 and continue as follows:

Algorithm 5. *Improved vector extraction.*

$$\begin{array}{l}\textit{for } i = 1 : n \\ \quad \sigma_i = \mathrm{fl}(\sigma_{i-1} + p_i) \\ \quad q_i = \sigma_i - \sigma_{i-1} \\ \quad p'_i = p_i - q_i \\ \textit{end for} \\ \tau = \sigma_n - \sigma_0\end{array}$$

The extraction in the for-loop is obviously the same as in the original vector extraction except that σ changes in each iteration. For suitable σ_0 one can show that both the computation of q_i and of p'_i are error-free, so that again $p_i = q_i + p'_i$ for all i. It follows

$$\sum p_i = \sum q_i + \sum p'_i.$$

The improvement of the new algorithm `FastAccSum` [11] is on the computation of $\sum q_i$. Using the fact that the computation of q_i by $q_i = \sigma_i - \sigma_{i-1}$ is error-free, we have a telescope sum:

$$q_1 + q_2 + \ldots + q_n = \sigma_1 - \sigma_0 + \sigma_2 - \sigma_1 + \ldots + \sigma_n - \sigma_{n-1} = \sigma_n - \sigma_0.$$

Summarizing we obtain

$$\sum p_i = \sigma_n - \sigma_0 + \sum p'_i.$$

Now the improved vector extraction requires only 3n floating-point operations, in contrast to the original vector extraction the computation of $\sum q_i$ comes practically free.

Of course, a number of details must be worked out; in particular the constants are chosen so that $\sigma_n - \sigma_0$ does also not cause a rounding error. This transforms the original sum of the p_i into $\tau + \sum p'_i$ as in `AccSum`.

In total this creates a new algorithm `FastAccSum` as presented in [11], and it is up to 25% faster than its origin `AccSum` ($3n$ compared to $4n$ flops in the inner loop).

In conclusion we presented new algorithms to compute accurate approximations of the sum of floating-point numbers. Details can be found in [7], [12], [13] and [11]. Over there also computing times for various compilers and architectures are presented showing that the new algorithms are the fastest known algorithms in terms of floating-point operations and in terms of executing time.

As has been explained, dot products can be transformed into sums of double length. Hence each summation algorithm implies an efficient way to compute accurate approximations of the dot product of two floating-point vectors.

The basic tool for all algorithms are error-free transformations. These well-known tools represent an interesting way to transform problems into simpler problems, the transformation performed without error.

Starting with [7] we see a renaissance of using error-free transformations (the term was coined in [7]) in various areas [2, 6, 5, 9, 14, 15]. We expect to see more of such applications in the near future.

References

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

[2] S. Graillat, P. Langlois, and N. Louvet, *Compensated Horner Scheme*, Tech. Report RR2005-02, Laboratoire LP2A, University of Perpignan, 2005.

[3] N. J. Higham, *Accuracy and Stability of Numerical Algorithms, Second Edition*, SIAM, Philadelphia, 2002.

[4] D.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, 1969.

[5] P. Langlois, *Accurate Algorithms in Floating Point Arithmetic*, Invited talk at the 12th GAMM–IMACS International Symposion on Scientific Computing, Computer Arithmetic and Validated Numerics, Duisburg, 26–29 September, 2006.

[6] P. Langlois and N. Louvet, *Solving Triangular Systems More Accurately and Efficiently*, Tech. Report RR2005-02, Laboratoire LP2A, University of Perpignan, 2005.

[7] T. Ogita, S.M. Rump, and S. Oishi. Accurate Sum and Dot Product. *SIAM Journal on Scientific Computing (SISC)*, 26(6):1955–1988, 2005.

[8] S. Oishi, K. Tanabe, T. Ogita and S.M. Rump. Convergence of Rump's method for inverting arbitrarily ill-conditioned matrices. *J. Comput. Appl. Math.*, 205(1):533–544, 2007.

[9] K. Ozaki, T. Ogita, S. M. Rump, and S. Oishi, *Fast and robust algorithm for geometric predicates using floating-point arithmetic*, Transactions of the Japan Society for Industrial and Applied Mathematics, 16 (2006), pp. 553–562.

[10] S.M. Rump. Inversion of extremely ill-conditioned matrices in floating-point. accepted for publication in JJIAM, 2008.

[11] S.M. Rump. Ultimately Fast Accurate Summation. submitted for publication, 2008.

[12] S.M. Rump, T. Ogita, and S. Oishi. Accurate Floating-point Summation Part I: Faithful Rounding. *SIAM Journal on Scientific Computing (SISC)*, 31(1):189–224, 2008.

[13] S.M. Rump, T. Ogita, and S. Oishi. Accurate Floating-point Summation Part II: Sign, K-fold Faithful and Rounding to Nearest. *SIAM Journal on Scientific Computing (SISC)*, 31(2):1269–1302, 2008.

[14] Y.-K. Zhu and W. Hayes, *Fast, guaranteed-accurate sums of many floating-point numbers*, in Proceedings of the 7th Conference on Real Numbers and Computers, G. Hanrot and P. Zimmermann, eds., 2006, pp. 11–22.

[15] Y.-K. Zhu, J.-H. Yong, and G.-Q. Zheng, *A new distillation algorithm for floating-point summation*, SIAM J. Sci. Comput., 26 (2005), pp. 2066–2078.