Tight interval inclusions with compensated algorithms

Stef Graillat & Fabienne Jézéquel

Sorbonne Université, Laboratoire d'Informatique de Paris 6 (LIP6), France

18th international symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2018)

Tokyo, Japan, 10-15 September 2018

Introduction

Exascale barrier broken in June 2018: $1.8\,10^{18}$ floating-point operations per second. (Oak Ridge National Laboratory, analysis of genomic data)

- Increasing power of current computers
  $\rightarrow$ GPU accelerators, Intel Xeon Phi processors, etc.
- Enable to solve more complex problems
  $\rightarrow$ Quantum field theory, supernova simulation, etc.
- A high number of floating-point operations performed
  $\rightarrow$ Each of them can lead to a rounding error

Introduction

Exascale barrier broken in June 2018: $1.8\,10^{18}$ floating-point operations per second. (Oak Ridge National Laboratory, analysis of genomic data)

- Increasing power of current computers
  $\rightarrow$ GPU accelerators, Intel Xeon Phi processors, etc.
- Enable to solve more complex problems
  $\rightarrow$ Quantum field theory, supernova simulation, etc.
- A high number of floating-point operations performed
  $\rightarrow$ Each of them can lead to a rounding error

$\Rightarrow$ Need for accuracy and validation

Key tools for accurate computation

- fixed length expansions libraries: double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
- arbitrary length expansions libraries: Priest, Shewchuk, Joldes-Muller-Popescu
- arbitrary precision libraries: ARPREC, MPFR, MPIR
- compensated algorithms (Kahan, Priest, Ogita-Rump-Oishi,...) based on EFTs (Error Free Transformations)

Key tools for accurate computation

- fixed length expansions libraries: double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
- arbitrary length expansions libraries: Priest, Shewchuk, Joldes-Muller-Popescu
- arbitrary precision libraries: ARPREC, MPFR, MPIR
- compensated algorithms (Kahan, Priest, Ogita-Rump-Oishi,...) based on EFTs (Error Free Transformations)

EFTs: properties and algorithms to compute the generated elementary rounding errors

Let $a, b \in \mathbb{F}$, for the basic operation $\circ \in \{+, -, \times\}$, with rounding to nearest,

$$a \circ b = \mathrm{fl}(a \circ b) + e \quad \text{with } e \in \mathbb{F}$$

Numerical validation with interval arithmetic

- Principle: replace numbers by intervals and compute.
- Fundamental theorem of interval arithmetic: the exact result belongs to the computed interval.
- No result is lost, the computed interval is guaranteed to contain every possible result.

Numerical validation with interval arithmetic

- Principle: replace numbers by intervals and compute.
- Fundamental theorem of interval arithmetic: the exact result belongs to the computed interval.
- No result is lost, the computed interval is guaranteed to contain every possible result.

How to compute tight interval inclusions
with compensated algorithms?

Numerical validation with interval arithmetic

- Principle: replace numbers by intervals and compute.
- Fundamental theorem of interval arithmetic: the exact result belongs to the computed interval.
- No result is lost, the computed interval is guaranteed to contain every possible result.

How to compute tight interval inclusions
with compensated algorithms?

Assume floating-point arithmetic adhering to IEEE 754 with rounding unit $\mathbf{u}$ (no underflow nor overflow).

Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Compensated algorithm for summation with directed rounding
4. Compensated dot product with directed rounding
5. Compensated Horner scheme with directed rounding

Outline

EFT for addition

$$x = a \oplus b \quad \Rightarrow \quad a + b = x + y \quad \text{with } y \in \mathbb{F}$$

Algorithm of Dekker (1971) and Knuth (1974)

Algorithm (EFT of the sum of 2 floating-point numbers with $|a| \geq |b|$)

function $[x, y] =$ `FastTwoSum`(a, b)

$x = a \oplus b$

$y = (a \ominus x) \oplus b$

Algorithm (EFT of the sum of 2 floating-point numbers)

function $[x, y] =$ `TwoSum`(a, b)

$x = a \oplus b$

$z = x \ominus a$

$y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

EFT for the product (1/3)

$$x = a \otimes b \quad \Rightarrow \quad a \times b = x + y \quad \text{with } y \in \mathbb{F}$$

Algorithm `TwoProduct` by Veltkamp and Dekker (1971)

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ non overlapping with } |y| \le |x|.$$

Algorithm (Error-free split of a floating-point number into two parts)

function $[x, y] = \texttt{Split}(a)$
  factor $= 2^s + 1$ % $\mathbf{u} = 2^{-p}$, $s = \lceil p/2 \rceil$
  $c = \texttt{factor} \otimes a$
  $x = c \ominus (c \ominus a)$
  $y = a \ominus x$

EFT for the product (2/3)

Algorithm (EFT of the product of 2 floating-point numbers)

function $[x, y] = \texttt{TwoProduct}(a, b)$

$x = a \otimes b$

$[a_1, a_2] = \texttt{Split}(a)$

$[b_1, b_2] = \texttt{Split}(b)$

$y = a_2 \otimes b_2 \ominus (((x \ominus a_1 \otimes b_1) \ominus a_2 \otimes b_1) \ominus a_1 \otimes b_2)$

EFT for the product (3/3)

$$x = a \otimes b \quad \Rightarrow \quad a \times b = x + y \quad \text{with } y \in \mathbb{F}$$

Given $a, b, c \in \mathbb{F}$,

- $\texttt{FMA}(a, b, c)$ is the nearest floating-point number to $a \times b + c$

Algorithm (EFT of the product of 2 floating-point numbers)

function $[x, y] = \texttt{TwoProdFMA}(a, b)$
  $x = a \otimes b$
  $y = \texttt{FMA}(a, b, -x)$

FMA is available for example on PowerPC, Itanium, Cell, Xeon Phi, AMD and Nvidia GPU, Intel (Haswell), AMD (Bulldozer) processors.

Outline

EFT for addition with directed rounding

$$x = \mathrm{fl}_*(a+b) \quad \Rightarrow \quad a+b = x+e \quad \text{but possibly } e \notin \mathbb{F}$$

Algorithm (EFT of the sum of 2 floating-point numbers with $|a| \geq |b|$)

function $[x, y] = \texttt{FastTwoSum}(a, b)$
  $x = \mathrm{fl}_*(a+b)$
  $y = \mathrm{fl}_*((a-x)+b)$

Proposition

We have $y = \mathrm{fl}_*(e)$ and so $|e-y| \leq 2\mathbf{u}|e|$. It yields $|e-y| \leq 4\mathbf{u}^2|x|$ and $|e-y| \leq 4\mathbf{u}^2|a+b|$. Moreover

- *if $* = \Delta$, $e \leq y$*
- *if $* = \nabla$, $y \leq e$*

EFT for addition with directed rounding

$$x = \mathrm{fl}_*(a+b) \;\Rightarrow\; a+b = x+e \quad \text{but possibly } e \notin \mathbb{F}$$

Algorithm (EFT of the sum of 2 floating-point numbers)

function $[x, y] = \texttt{TwoSum}(a, b)$
  $x = \mathrm{fl}_*(a+b)$
  $z = \mathrm{fl}_*(x-a)$
  $y = \mathrm{fl}_*((a-(x-z)) + (b-z))$

Proposition

We have $|e-y| \le 4\mathbf{u}^2|a+b|$ *and* $|e-y| \le 4\mathbf{u}^2|x|$. *Moreover*

- *if* $* = \Delta$, $e \le y$
- *if* $* = \nabla$, $y \le e$

EFT for the product with directed rounding

$$x = \mathrm{fl}_*(a \times b) \quad \Rightarrow \quad a \times b = x + y \quad \text{with } y \in \mathbb{F}$$

Given $a, b, c \in \mathbb{F}$,

- $\texttt{FMA}(a, b, c)$ is the nearest floating-point number to $a \times b + c$

Algorithm (EFT of the product of 2 floating-point numbers)

function $[x, y] = \texttt{TwoProdFMA}(a, b)$
  $x = \mathrm{fl}_*(a \times b)$
  $y = \texttt{FMA}(a, b, -x)$

EFT for the product with directed rounding

$$a = x + y \quad \text{and} \quad x \text{ and } y \text{ non overlapping with } |y| \le |x|$$

Algorithm (Error-free split of a floating-point number into two parts)

function $[x, y] = \texttt{Split}(a)$
  factor $= 2^s + 1$ % $\mathbf{u} = 2^{-p}$, $s = \lceil p/2 \rceil$
  $c = \mathrm{fl}_*(\texttt{factor} \times a)$
  $x = \mathrm{fl}_*(c - (c - a))$
  $y = \mathrm{fl}_*(a - x)$

Proposition

We have $a = x + y$. Moreover,

- *the significand of x fits in $p - s$ bits;*
- *the significand of y fits in s bits.*

EFT for the product with directed rounding

$$x = \mathrm{fl}_*(a \times b) \quad \Rightarrow \quad a \times b = x + e \quad \text{with } e \in \mathbb{F}$$

Algorithm (EFT of the product of 2 floating-point numbers)

function $[x, y] = \texttt{TwoProduct}(a, b)$
$x = \mathrm{fl}_*(a \times b)$
$[a_1, a_2] = \texttt{Split}(a)$; $[b_1, b_2] = \texttt{Split}(b)$
$y = \mathrm{fl}_*(a_2 \times b_2 - (((x - a_1 \times b_1) - a_2 \times b_1) - a_1 \times b_2))$

Proposition

We have $|e - y| \leq 8\mathbf{u}^2 |a \times b|$ *and* $|e - y| \leq 8\mathbf{u}^2 |x|$. *Moreover*

- *if* $* = \Delta$, $e \leq y$
- *if* $* = \nabla$, $y \leq e$

Outline

Compensated algorithm for summation

Let $p = \{p_i\}$ be a vector of n floating-point numbers.

Algorithm (Ogita, Rump, Oishi (2005))

function $\texttt{res} = \texttt{CompSum}(p)$
  $\pi_1 = p_1$; $\sigma_1 = 0$
  for $i = 2 : n$
    $[\pi_i, q_i] = \texttt{TwoSum}(\pi_{i-1}, p_i)$
    $\sigma_i = \mathrm{fl}_*(\sigma_{i-1} + q_i)$
  $\texttt{res} = \mathrm{fl}_*(\pi_n + \sigma_n)$

Proposition

Let us suppose CompSum *is applied, with directed rounding, to* $p_i \in \mathbb{F}$, $1 \leq i \leq n$. *Let* $s := \sum p_i$ *and* $S := \sum |p_i|$. *If* $n\mathbf{u} < \frac{1}{2}$, *then*

$$|\texttt{res} - s| \leq 2\mathbf{u}|s| + 2(1 + 2\mathbf{u})\gamma_n^2(2\mathbf{u})S \quad \text{with} \quad \gamma_n(\mathbf{u}) = \frac{n\mathbf{u}}{1 - n\mathbf{u}}.$$

Compensated algorithm for summation

Algorithm (Tight inclusion using INTLAB)

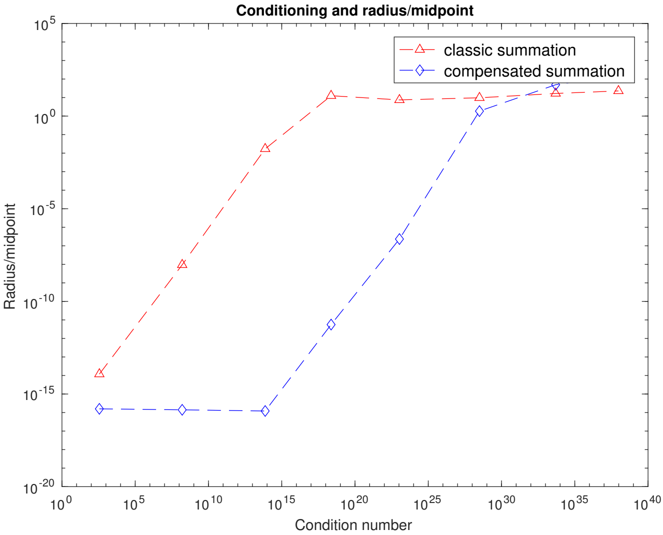
```
setround(-1)
Sinf = CompSump(p)
setround(1)
Ssup = CompSump(p)
```

Proposition

Let $p = \{p_i\}$ be a vector of n floating-point numbers. Then we have

$$\texttt{Sinf} \le \sum_{i=1}^{n} p_i \le \texttt{Ssup}.$$

Numerical experiments



Outline

Compensated dot product

Algorithm (Ogita, Rump and Oishi 2005)

function $\texttt{res} = \texttt{CompDot}(x, y)$
  $[p, s] = \texttt{TwoProduct}(x_1, y_1)$
  for $i = 2 : n$
    $[h, r] = \texttt{TwoProduct}(x_i, y_i)$
    $[p, q] = \texttt{TwoSum}(p, h)$
    $s = \mathrm{fl}_*(s + (q + r))$
  end
  $\texttt{res} = \mathrm{fl}_*(p + s)$

Proposition

Let $x_i, y_i \in \mathbb{F}$ $(1 \leq i \leq n)$ and $\texttt{res}$ *the result computed by* `CompDot` *with directed rounding. If $(n+1)\mathbf{u} < \frac{1}{2}$, then,*

$$|\texttt{res} - x^T y| \leq 2\mathbf{u}|x^T y| + 2\gamma_{n+1}^2(2\mathbf{u})|x^T||y|.$$

Compensated dot product

Algorithm (Tight inclusion using INTLAB)

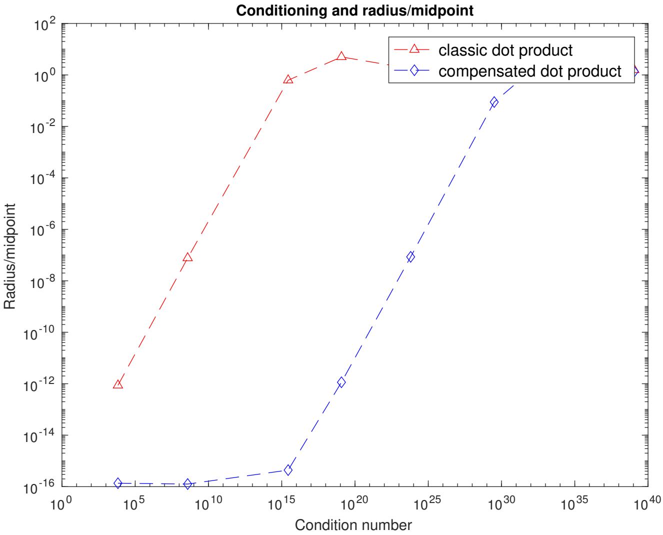
```
setround(-1)
Dinf = CompDot(x,y)
setround(1)
Dsup = CompDot(x,y)
```

Proposition

Let $x_i, y_i \in \mathbb{F}$ $(1 \leq i \leq n)$ *be given. Then we have*

$$\texttt{Dinf} \leq x^T y \leq \texttt{Dsup}.$$

Numerical experiments



Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Compensated algorithm for summation with directed rounding
4. Compensated dot product with directed rounding
5. Compensated Horner scheme with directed rounding

Compensated Horner scheme

Let $p(x) = \sum_{i=0}^{n} a_i x^i$ with $x, a_i \in \mathbb{F}$

Algorithm (Graillat, Langlois, Louvet, 2009)

function $\texttt{res} = \texttt{CompHorner}(p, x)$

$s_n = a_n$

$r_n = 0$

for $i = n - 1 : -1 : 0$

$[p_i, \pi_i] = \texttt{TwoProduct}(s_{i+1}, x)$

$[s_i, \sigma_i] = \texttt{TwoSum}(p_i, a_i)$

$r_i = \mathrm{fl}_*(r_{i+1} \times x + (\pi_i + \sigma_i))$

end

$\texttt{res} = \mathrm{fl}_*(s_0 + r_0)$

Compensated Horner scheme

Theorem

Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x. With directed rounding, the forward error in the compensated Horner algorithm is such that

$$|\texttt{CompHorner}(p, x) - p(x)| \leq 2\mathbf{u}|p(x)| + 2\gamma_{2n+1}(2\mathbf{u})^2\widetilde{p}(|x|),$$

with $\widetilde{p}(x) = \sum_{i=0}^{n} |a_i|x^i$.

Compensated Horner scheme

Algorithm ($x \geq 0$, Tight inclusion using INTLAB)

```
setround(-1)
Einf = CompHorner(p,x)
setround(1)
Esup = CompHorner(p,x)
```

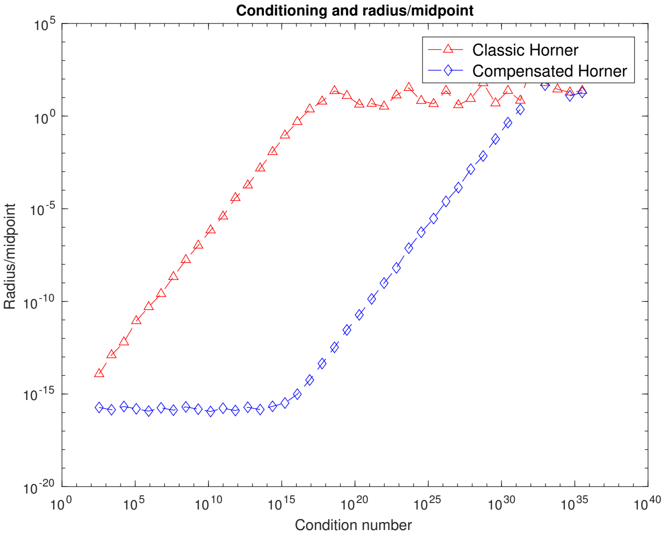
If $x \leq 0$, $\texttt{CompHorner}(\bar{\texttt{p}}, -\texttt{x})$ is computed with $\bar{p}(x) = \sum_{i=0}^{n} a_i(-1)^i x^i$.

Proposition

Consider a polynomial p of degree n with floating-point coefficients, and a floating-point value x.

$$\texttt{Einf} \leq p(x) \leq \texttt{Esup}.$$

Numerical experiments



Conclusion and future work

Conclusion

- Compensated methods are a fast way to get accurate results
- They can be used efficiently with interval arithmetic to obtain certified results with finite precision

Future work

- Interval computations with K-fold compensated algorithms using Priest's EFT and FMA

References I

[1] S. Boldo, S. Graillat, and J.-M. Muller.
On the robustness of the 2Sum and Fast2Sum algorithms.
ACM Trans. Math. Softw., 44(1):4:1–4:14, July 2017.

[2] S. Graillat, F. Jézéquel, and R. Picot.
Numerical validation of compensated summation algorithms with stochastic arithmetic.
Electronic Notes in Theoretical Computer Science, 317:55–69, 2015.

[3] S. Graillat, F. Jézéquel, and R. Picot.
Numerical validation of compensated algorithms with stochastic arithmetic.
Applied Mathematics and Computation, 329:339 – 363, 2018.

References II

[4] S. Graillat, Ph. Langlois, and N. Louvet.
Algorithms for accurate, validated and fast polynomial evaluation.
Japan J. Indust. Appl. Math., 2-3(26):191–214, 2009.

[5] T. Ogita, S. M. Rump, and S. Oishi.
Accurate sum and dot product.
SIAM Journal on Scientific Computing, 26(6):1955–1988, 2005.

[6] D.M. Priest.
On Properties of Floating Point Arithmetics: Numerical Stability and the Cost of Accurate Computations.
PhD thesis, Mathematics Department, University of California, Berkeley, CA, USA, November 1992.

Thank you for your attention