Accurate and validated numerical computing

Fabienne Jézéquel

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

Workshop on Large-scale Parallel Numerical Computing Technology
RIKEN Center for Computational Science, Kobe, Japan, 6-8 June 2019

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
  - $\longrightarrow$ GPU accelerators, Intel Xeon Phi processors, etc.
- Enable to solve more complex problems
  - $\longrightarrow$ Quantum field theory, supernova simulation, etc.
- A high number of floating-point operations performed
  - $\longrightarrow$ 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

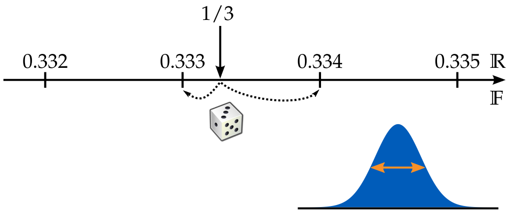
Let $a, b \in \mathbb{F}$, for the basic operation $\circ \in \{+, -, \times\}$, with rounding to nearest,

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

Numerical validation: 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.
- Some implementations:
  - INTLAB [Rump]
    `http://www.ti3.tu-harburg.de/intlab`
  - GNU octave interval package [Heimlich]
    `http://octave.sourceforge.net/interval`
  - Boost interval arithmetic library [Brönnimann et al., 2006]
  - C-XSC [Hofschuster et al., 2004]
  - filib++ [Lerch et al, 2006]

Numerical validation: Discrete Stochastic Arithmetic (DSA) [Vignes, 2004]



- each operation executed 3 times with a random rounding mode
- number of correct digits in the results estimated using Student's test with the probability 95%
- estimation may be invalid if both operands in a multiplication or a divisor are not significant.
  $\Rightarrow$ control of multiplications and divisions: *self-validation* of DSA.
- in DSA rounding errors are assumed centered.
  even if they are not rigorously centered, the accuracy estimation can be considered correct up to 1 digit.

Implementation of DSA

- CADNA: for programs in single and/or double precision
  `http://cadna.lip6.fr`
- SAM: for arbitrary precision programs (based on MPFR)
  `http://www-pequan.lip6.fr/~jezequel/SAM`

- estimate accuracy and detect numerical instabilities
- provide stochastic types (3 classic type variables and 1 integer)
- all operators and mathematical functions overloaded
  $\Rightarrow$ few modifications in user programs

In this talk...

Results established for directed rounding will be applied to

- interval arithmetic
- discrete stochastic arithmetic.

In this talk...

Results established for directed rounding will be applied to

- interval arithmetic
- discrete stochastic arithmetic.

Notations:

- We assume floating-point arithmetic adhering to IEEE 754 with rounding unit $\mathbf{u}$ (no underflow nor overflow).
- Let
$$\gamma_n(\mathbf{u}) = \frac{n\mathbf{u}}{1 - n\mathbf{u}}.$$

Outline of this talk

I. Compensated algorithms and numerical validation

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
4. Numerical validation of compensated algorithms with DSA

II. Numerical validation of quadruple or arbitrary precision programs with DSA

1. Implementation of DSA in quadruple precision
2. Implementation of DSA in arbitrary precision
3. Application: computation of multiple roots of polynomials

I. Compensated algorithms and numerical validation

Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

EFT for the addition with rounding to nearest

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

Algorithm of Dekker (1971) and Knuth (1974)

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

function $[x, y] = \texttt{FastTwoSum}(a, b)$
$\quad x = a \oplus b$
$\quad y = (a \ominus x) \oplus b$

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

function $[x, y] = \texttt{TwoSum}(a, b)$
$\quad x = a \oplus b$
$\quad z = x \ominus a$
$\quad y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

EFT for the product with rounding to nearest

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

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

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

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

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

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

Outline

EFT for the 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 the 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)

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

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

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

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

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

Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

Compensated summation with directed rounding

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*

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

Tight inclusions with compensated summation

Algorithm (Tight inclusion using INTLAB)

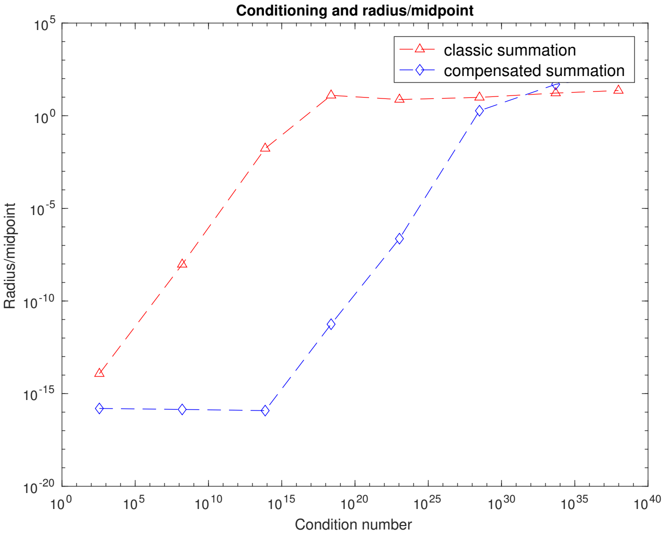
```
setround(-1)
Sinf = CompSum(p)
setround(1)
Ssup = CompSum(p)
```

Proposition

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

$$\mathtt{Sinf} \leq \sum_{i=1}^{n} p_i \leq \mathtt{Ssup}.$$

Numerical experiments



Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

Compensated dot product

Algorithm (Ogita, Rump and Oishi 2005)

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

Proposition

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

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

Tight inclusions with compensated dot product

Algorithm (Tight inclusion using INTLAB)

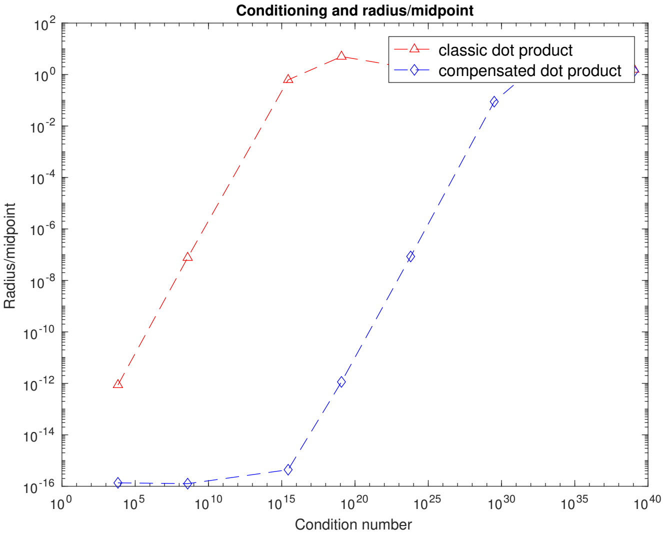
```
setround(-1)
Dinf = CompDot(x,y)
setround(1)
Dsup = CompDot(x,y)
```

Proposition

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

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

Numerical experiments



Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision 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 $i = n-1 : -1 : 0$
  - $[p_i, \pi_i] = \texttt{TwoProdFMA}(s_{i+1}, x)$
  - $[s_i, \sigma_i] = \texttt{TwoSum}(p_i, a_i)$
  - $r_i = \mathrm{fl}_*(r_{i+1} \times x + (\pi_i + \sigma_i))$
- 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

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

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

Tight inclusions - compensated Horner scheme

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

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

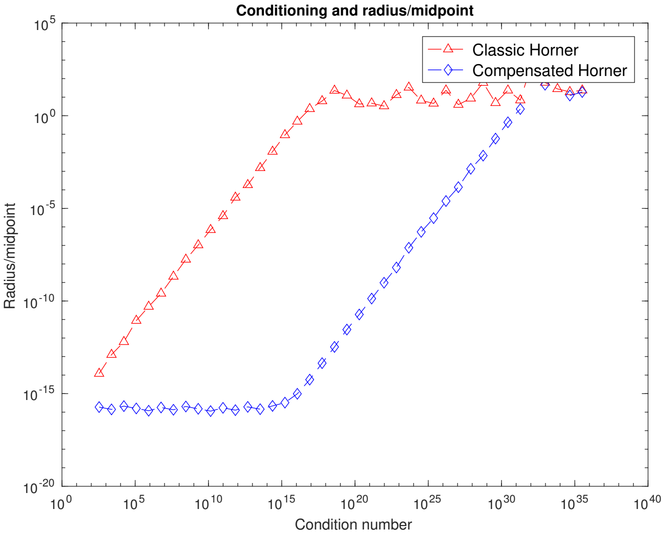
If $x \leq 0$, $\mathtt{CompHorner}(\bar{\mathtt{p}}, -\mathtt{x})$ is computed
with $\bar{p}(x) = \sum_{i=0}^{n} a_i (-1)^i x^i$.

Proposition

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

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

Numerical experiments



Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

Compensated algorithms with DSA

With the random rounding mode, EFTs are no more exact.

However thanks to the error bounds obtained with directed rounding,

- CADNA can be used to validate results of compensated algorithms.
- compensated algorithms can be used in CADNA codes.

For classic algorithms and their compensated versions, we compare:

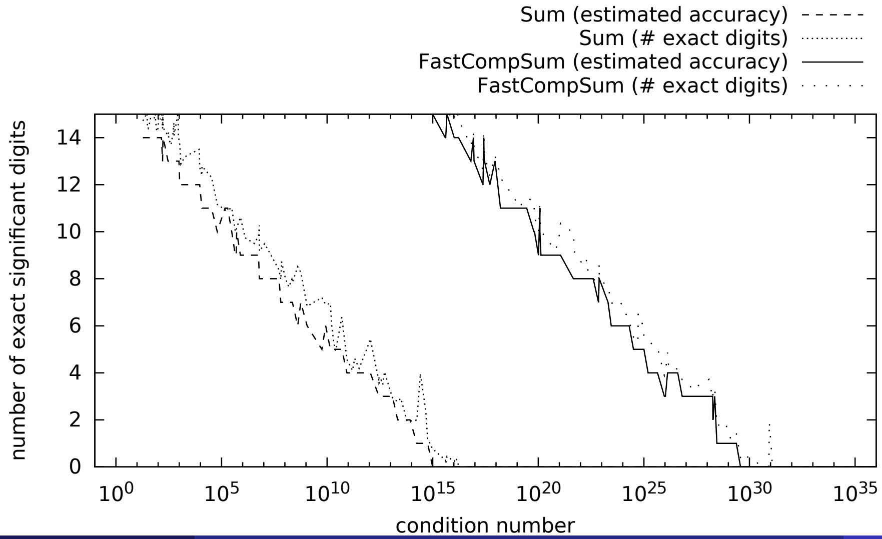
- the accuracy estimated by CADNA
- the accuracy d evaluated from the exact results

R: result computed with CADNA
R_{exact}: result computed symbolically

$$\begin{aligned} \text{If } R_{exact} \neq 0, \quad d &= -\log_{10}\left|\frac{R - R_{exact}}{R_{exact}}\right|, \\ \text{otherwise} \quad d &= -\log_{10}|R|. \end{aligned}$$

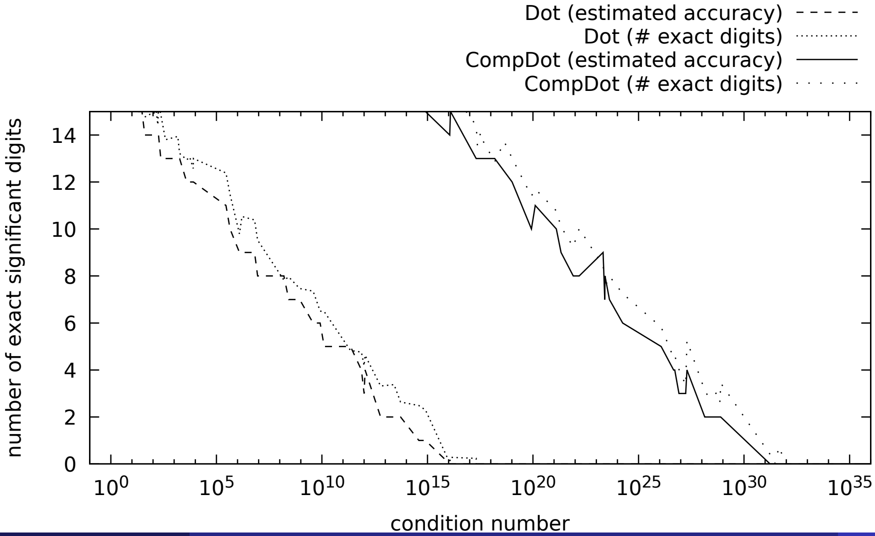
Compensated summation using DSA

Accuracy (estimated by CADNA and computed from the exact results) using the `Sum` and the `FastCompSum` algorithms



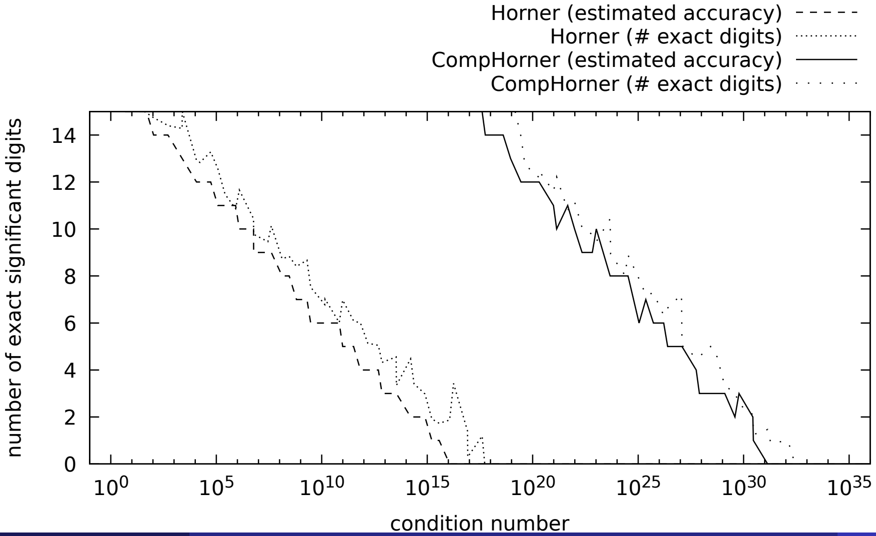
Compensated dot product using DSA

Accuracy (estimated by CADNA and computed from the exact results) using the `Dot` and the `CompDot` algorithms



Compensated Horner scheme using DSA

Accuracy (estimated by CADNA and computed from the exact results) using the `Horner` and the `CompHorner` algorithms



Outline

1. Error-free transformations (EFT) with rounding to nearest

2. Error-free transformations (EFT) with directed rounding

3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding

4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

EFT for the addition with any rounding mode

From a and b, Priest algorithm (1992) returns x and y such that $a + b = x + y$.

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

function $[x, y] = \texttt{PriestTwoSum}(a, b)$
  if $|b| > |a|$
    exchange a and b
  endif
  $x = fl_*(a + b)$; $e = fl_*(x - a)$
  $g = fl_*(x - e)$; $h = fl_*(g - a)$
  $f = fl_*(b - h)$; $y = fl_*(f - e)$
  if $fl_*(y + e) \neq f$
    $x = a$; $y = b$
  endif

Compensated summation with directed rounding

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

Algorithm

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

Proposition

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

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

Summation as in K-fold precision

with directed rounding

`SumK` based on `TwoSum` introduced by Ogita, Rump & Oishi (2005)

Algorithm (Summation in K-fold working precision)

function $\texttt{res} = \texttt{SumK}(p, K)$
  for $k = 1 : K - 1$
    for $i = 2 : n$
      $[p_i, p_{i-1}] = \texttt{PriestTwoSum}(p_i, p_{i-1})$
  $\texttt{res} = \mathrm{fl}_*(\sum_{i=1}^{n} p_i)$

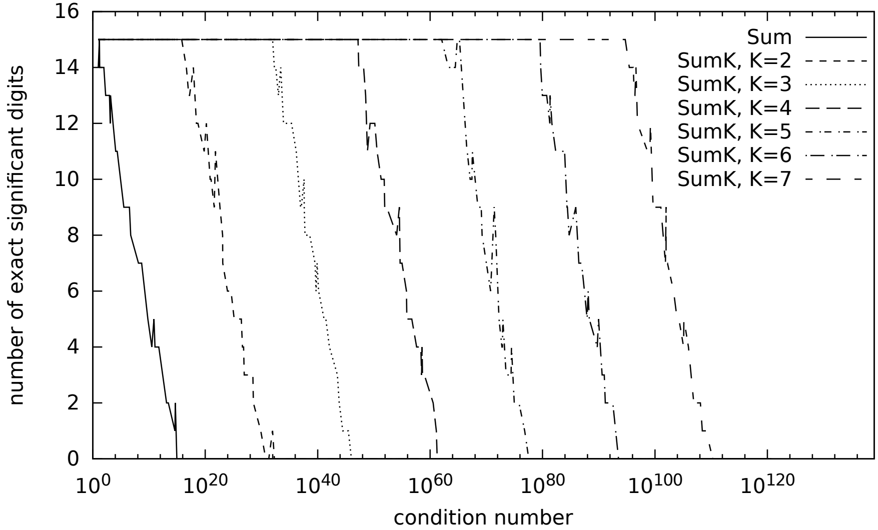
Proposition

Let us suppose `SumK` *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* $8n\mathbf{u} \leq 1$, *then*

$$|\texttt{res} - s| \leq \left(2\mathbf{u} + 3\gamma_{n-1}^2(2\mathbf{u})\right)|s| + \gamma_{2n-2}^K(2\mathbf{u})S.$$

Summation as in K-fold precision using DSA

Accuracy estimated by CADNA using the `Sum` and the `SumK` algorithms



Outline

1. Error-free transformations (EFT) with rounding to nearest
2. Error-free transformations (EFT) with directed rounding
3. Tight interval inclusions with compensated algorithms
   - Compensated algorithm for summation with directed rounding
   - Compensated dot product with directed rounding
   - Compensated Horner scheme with directed rounding
4. Numerical validation of compensated algorithms with DSA
   - Compensated algorithms for summation, dot product and Horner scheme with DSA
   - Summation as in K-fold precision with directed rounding
   - Dot product as in K-fold precision with directed rounding

Dot product as in K-fold precision with directed rounding

`DotK` based on `TwoSum` and `TwoProd` (EFT with rounding to nearest) introduced by Ogita, Rump & Oishi (2005)

Algorithm (Dot product in K-fold working precision)

function $\mathtt{res} = \mathtt{DotK}(x, y, K)$
  $[p, r_1] = \mathtt{TwoProdFMA}(x_1, y_1)$
  for $i = 2 : n$
    $[h, r_i] = \mathtt{TwoProdFMA}(x_i, y_i)$; $[p, r_{n+i-1}] = \mathtt{PriestTwoSum}(p, h)$
  $r_{2n} = p$
  $\mathtt{res} = \mathtt{SumK}(r, K-1)$

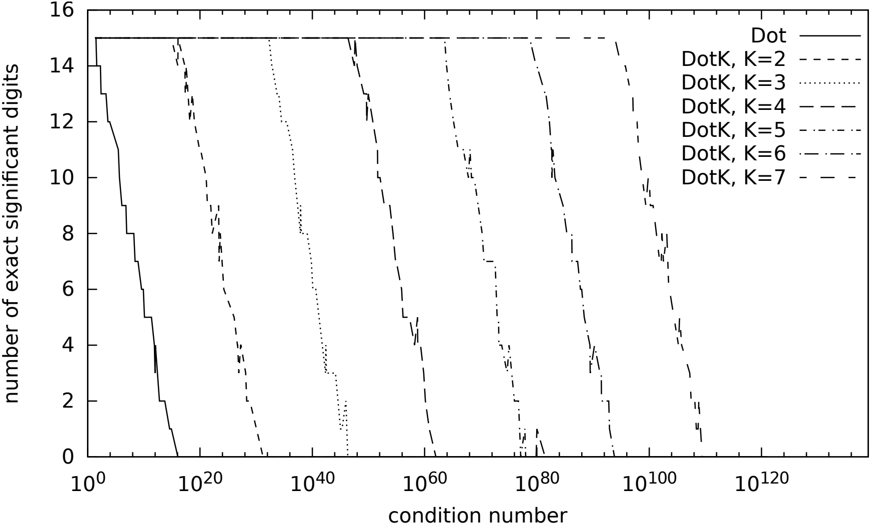
Proposition

Let $x_i, y_i \in \mathbb{F}$ $(1 \leq i \leq n)$ and `res` *the result computed by* `DotK` *with directed rounding. If $16n\mathbf{u} \leq 1$, then*

$$|\mathtt{res} - x^T y| \leq \left(\mathbf{u} + 2\gamma_{4n-2}^2(\mathbf{u})\right)|x^T y| + \gamma_{4n-2}^K(\mathbf{u})|x^T||y|.$$

Dot product as in K-fold precision using DSA

Accuracy estimated by CADNA using the `Dot` and the `DotK` algorithms



Conclusion and future work

Conclusion

- Compensated algorithms are a fast way to get accurate results
- Compensated algorithms & interval arithmetic
  → certified results with finite precision
- Compensated algorithms & stochastic arithmetic
  → results with accuracy estimation

Future work

- Error analysis of TwoProd with directed rounding
- Interval computation with K-fold compensated algorithms

II. Quadruple and arbitrary precision with DSA

Introduction

- Simulation programs usually in single (*binary32*) or double (*binary64*) precision
- Quadruple precision (*binary128*) or arbitrary precision sometimes required, for instance:
  - computation of chaotic sequences
  - approximation of multiple roots of polynomials

How to estimate efficiently rounding errors
in quadruple/arbitrary precision codes?

Probabilistic approach

- uses a random rounding mode
- DSA (Discrete Stochastic Arithmetic):
  - estimates the number of exact significant digits of any computed result
  - implemented in the CADNA library

Overview

- Extension of DSA to quadruple precision
- Extension of DSA to arbitrary precision
- Application: computation of multiple roots of polynomials

Quadruple precision arithmetic

binary128 format

- a sign bit, a 15-bit long exponent, a 112-bit long mantissa
  actual mantissa precision: 113 bits
- in GCC, bit field structure:
  - sign bit
  - 15-bit long integer for the exponent
  - 48-bit long integer for the high part of the mantissa
  - 64-bit long integer for its low part.

double-double format

- $a = (a_h, a_l)$ with $a = a_h + a_l$, $|a_l| \leq 2^{-53}|a_h|$
- rounding not as specified in the IEEE 754 standard
- the QD library [Hida, Li & Bailey, 2008]
  - several implementations of *double-double* operations
  - for $+$ and $/$, *sloppy* version: performs better with a possibly higher error.

Performance comparison of quadruple precision programs

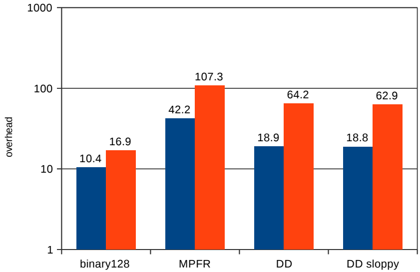
Comparison of:

- *binary128*
- MPFR with 113-bit mantissa length
  `http://www.mpfr.org`
- *double-double* implementations from QD
  `http://www.davidhbailey.com/dhbsoftware`

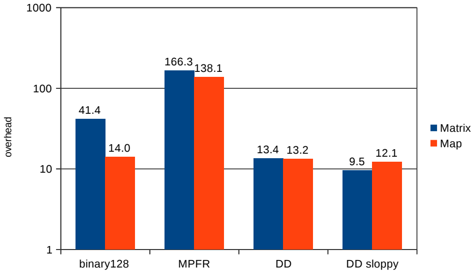
Benchmarks:

- *Matrix*: naive multiplication of two square matrices of size 1,000.
- *Map*: computes the sequence
  - $U_0 = 1.1$
  - for $i = 1,\dots, n$, $U_i = (0.1 \times U_{i-1} - (1/3 + U_{i-1})^2)/(1 - U_{i-1})^3$
    with $n = 128,000,000$.

Performance ratio w.r.t. double precision (*binary64*)



(a) GCC with O0



(b) GCC with O3

MPFR (3.1.1) costly, although improvements expected with version 4.
with O0:

- *binary128* performs the best
- low performance gain with DD sloppy

with O3:

- Matrix: performance ratio *binary128/binary64* higher than with O0
- DD better than *binary128*, but does not adhere to IEEE 754

same trends with ICC

Extension of CADNA to quadruple precision

- new *binary128* type:
  `__float128` (GCC), `_Quad` (ICC) $\Rightarrow$ `float128`
- new CADNA type:
  `float128_st` (3 `float128` and 1 integer)
- efficient rounding mode change:
  - implicit change of the rounding mode thanks to
    $a \oplus_{+\infty} b = -(-a \oplus_{-\infty} -b)$ (similarly for $\ominus$)
    $a \otimes_{+\infty} b = -(a \otimes_{-\infty} -b)$ (similarly for $\oslash$)
    $\bigcirc_{+\infty}$ (resp. $\bigcirc_{-\infty}$): floating-point operation rounded $\rightarrow +\infty$ (resp. $-\infty$)
  - bit flip of `float128` numbers
- overloading of arithmetic operators and mathematical functions

Performance of CADNA in quadruple precision

CADNA overhead w.r.t. classic floating-point computation (GCC):

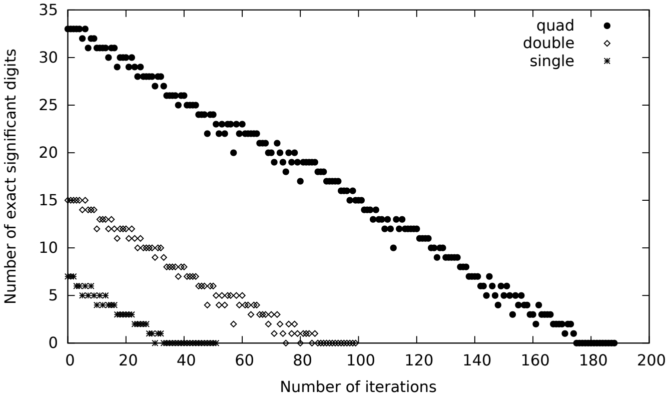
		no instability	self-validation	all instabilities
single	matrix	15	16	34
	map	10	15	20
double	matrix	20	22	35
	map	11	14	20
quadruple	matrix	5.0	5.4	21
	map	7.8	12	19

- lower overheads in quadruple precision
- instability detection cost not particularly higher in quadruple precision

Numerical experiment: Hénon map [Hénon, 1976]

maps a point $(x_i, y_i) \in \mathbb{R}^2$ to a new point defined by $x_{i+1} = 1 + y_i - a\,x_i^2$ and $y_{i+1} = b\,x_i$.

Accuracy estimated by CADNA of coordinates x_i of the Hénon map with $a = 1.4$, $b = 0.3$, $x_0 = 1$, and $y_0 = 0$:



Points (x_i, y_i) of the Hénon map computed using CADNA with $a = 1.4$, $b = 0.3$, $x_0 = 1$, and $y_0 = 0$ (exact digits displayed, @.0 if numerical noise):

iteration	precision	point (x_i, y_i) computed using CADNA	
1	single	x_i	-0.3999999E+000
		y_i	0.3000000E+000
1	double	x_i	-0.399999999999999E+000
		y_i	0.300000000000000E+000
1	quad	x_i	-0.399999999999999999999999999999999E+000
		y_i	0.300000000000000000000000000000000E+000
30	single	x_i	@.0
		y_i	0.2E+000
30	double	x_i	-0.13848191E+000
		y_i	0.2856319104E+000
30	quad	x_i	-0.138481919146792462486489312E+000
		y_i	0.2856319104003007180980589904E+000
75	double	x_i	@.0
		y_i	-0.1E+000
75	quad	x_i	0.115649947336564503E+000
		y_i	-0.1839980672458806840E+000
175	quad	x_i	@.0
		y_i	-0.2E+000

The need for arbitrary precision

Floating-point arithmetic precision

- IEEE single precision: 32 bits (24-bit mantissa)
- IEEE double precision: 64 bits (53-bit mantissa)
- IEEE quadruple precision: 128 bits (113-bit mantissa)

Because of round-off errors, some problems must be solved with a longer floating-point format.
`http://crd.lbl.gov/~dhbailey/dhbpapers/hpmpd.pdf`

⤇ Arbitrary precision libraries

- ARPREC
  `http://crd.lbl.gov/~dhbailey/mpdist`
- MPFR
  `http://www.mpfr.org`

Numerical validation & arbitrary precision

In arbitrary precision, round-off errors still occur...
and require to be controlled!

MPFI: interval arithmetic in arbitrary precision, based on MPFR
`http://mpfi.gforge.inria.fr`
interval arithmetic not well suited for the validation of huge applications ☹

CADNA: stochastic arithmetic
`http://cadna.lip6.fr`
used for the validation of real-life applications ☺
in single, double or quadruple precision

$\Longmapsto$ SAM: Stochastic Arithmetic in Multiprecision
`http://www-pequan.lip6.fr/~jezequel/SAM`

The SAM library

The SAM library implements in arbitrary precision the features of DSA:

- the stochastic types
- the concept of computational zero
- the stochastic operators.

The particularity of SAM (compared to CADNA) is the arbitrary precision of stochastic variables.

Using the SAM library

- The SAM library is written in C++ and is based on MPFR.
- All operators are overloaded
  $\Rightarrow$ for a program in C++ to be used with SAM, only a few modifications are needed.
- Classical variables $\rightarrow$ stochastic variables (of `mp_st` type) consisting of
  - three variables of MPFR type
  - one integer variable to store the accuracy.

How to implement SAM?

- declaration of the SAM library for the compiler
  `#include "sam.h"`
- initialization of the SAM library
  `sam_init(nb_instabilities, nb_bits);`
- substitution of `float` or `double` by the stochastic type `mp_st` in variable declarations
- change of output statements to print stochastic results with their accuracy,
  only the significant digits not affected by round-off errors are displayed
- termination of the SAM library
  `sam_end();`

Example of SAM code

$$f(x,y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}$$

is computed with $x = 77617$, $y = 33096$.

S. Rump, 1988

```
#include "sam.h"
#include <stdio.h>
int main() {
   sam_init(-1,122);
   mp_st x = 77617; mp_st y = 33096; mp_st res;
   res=333.75*y*y*y*y*y*y+x*x*(11*x*x*y*y-y*y*y*y*y*y
      -121*y*y*y*y-2.0)+5.5*y*y*y*y*y*y*y*y+x/(2*y);
   printf("res=%s\n",strp(res));
   sam_end();
}
```

Output of the SAM code

Using SAM with 122-bit mantissa length, one obtains:

```
Self-validation detection:  ON
Mathematical instabilities detection:  ON
Branching instabilities detection:  ON
Intrinsic instabilities detection:  ON
Cancellation instabilities detection:  ON
--------------
res=-0.827396059946821368141165095479816292
--------------
No instability detected
```

Computation of $f(77617, 33096)$

single precision	2.571784e+29
double precision	1.1726039400531
extended precision	1.172603940053178
Variable precision interval arithmetic	$[-0.827396059946821368141165095479816292005,$ $-0.827396059946821368141165095479816291986]$
SAM, 121 bits	@.0
SAM, 122 bits	$-0.827396059946821368141165095479816292$

Application of SAM to chaotic dynamical systems

Logistic iteration:

$$x_{n+1} = ax_n(1 - x_n) \text{ with } a > 0 \text{ and } 0 < x_0 < 1$$

- $a < 3$: $\forall x_0$, this sequence converges to a unique fixed point.
- $3 \leq a \leq 3.57$: $\forall x_0$, this sequence is periodic, the periodicity depending only on a. Furthermore the periodicity is multiplied by 2 for some values of a called "bifurcations".
- $3.57 < a < 4$: this sequence is usually chaotic, but there are certain isolated values of a that appear to show periodic behavior.
- $a \geq 4$: the values eventually leave the interval [0,1] and diverge for almost all initial values.

Logistic map

The logistic map has been computed with $x_0 = 0.6$ using SAM and MPFI

- In stochastic arithmetic, iterations have been performed until the current iterate is a computational zero, *i.e.* all its digits are affected by round-off errors.
- In interval arithmetic, iterations have been performed until the two bounds of the interval have no common significant digit.

Comparison of SAM and MPFI - I

Number N of iterations performed with SAM and MPFI, for $x_{n+1} = ax_n(1 - x_n)$ with $x_0 = 0.6$.

a		# bits	N
3.575	SAM	24	142
	SAM	53	372
	SAM	100	802
	SAM	200	1554
	SAM	2000	15912
	MPFI	24	12
	MPFI	53	27
	MPFI	100	53
	MPFI	200	108
	MPFI	2000	1087
3.6	SAM	24	62
	SAM	53	152
	SAM	100	338
	SAM	200	724
	MPFI	24	12
	MPFI	53	27
	MPFI	100	53
	MPFI	200	107

Comparison of SAM and MPFI - II

Number N of iterations performed with SAM and MPFI, $x_{n+1} = -a(x_n - \frac{1}{2})^2 + \frac{a}{4}$

a		# bits	N
3.575	SAM	24	156
	SAM	53	362
	SAM	100	738
	SAM	200	1558
	SAM	2000	15958
	MPFI	24	93
	MPFI	53	303
	MPFI	100	707
	MPFI	200	1517
	MPFI	2000	15865
3.6	SAM	24	56
	SAM	53	156
	SAM	100	344
	SAM	200	730
	MPFI	24	49
	MPFI	53	143
	MPFI	100	329
	MPFI	200	713

Numerical experiment: multiple roots of polynomials

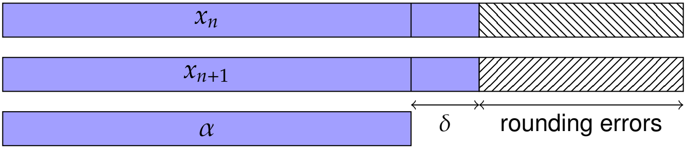
Newton's method to approximate a root α of a function f:

- compute the sequence $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$.
- classic stopping criterion: $|x_{n+1} - x_n| < \varepsilon$ or $\left|\frac{x_{n+1}-x_n}{x_{n+1}}\right| < \varepsilon$ if $x_{n+1} \neq 0$.

Approximation of a root α with multiplicity $m \geq 2$

using Newton's method

- Optimal stopping criterion: $x_n - x_{n+1}$ not significant (numerical noise)
- Then the digits of x_{n+1} which are not affected by rounding errors are in common with α, up to $\delta = \lceil \log_{10}(m-1) \rceil$. [Graillat et al., 2016]



Numerical experiment with CADNA

We compute using Newton's method approximations of the root α of $P_m(x) = (x-1)^m$ and for each approximation:

- #digits not affected by rounding errors estimated by CADNA
- #digits in common with the exact root α: $-\log_{10}\left|\frac{\alpha_{computed}-\alpha}{\alpha}\right|$

precision	$m = 6$, $\delta = 1$		$m = 18$, $\delta = 2$	
	CADNA	exact	CADNA	exact
single	2	1.0	1	0.5
double	3	2.4	1	0.7
quad	7	5.5	3	1.6

As expected, the accuracy estimated by CADNA is correct, up to δ or $\delta + 1$.
(recall: the accuracy estimation by CADNA can be considered correct up to 1 digit).

Numerical experiment with SAM

We compute using Newton's method approximations of the root α of $P(x) = (3x-1)^{100}$ ($\delta = 2$) and for each approximation:

- #digits not affected by rounding errors estimated by SAM
- #digits in common with the exact root α: $\lfloor -\log_{10} \left| \frac{\alpha_{computed} - \alpha}{\alpha} \right| \rfloor$

precision		#digits	
bits	digits	SAM	exact
200	60	1	0
500	150	3	1
1000	301	4	2
5000	1505	17	14
10000	3010	31	29

As expected, the accuracy estimated by SAM is correct, up to δ or $\delta + 1$. (recall: the accuracy estimation by SAM can be considered correct up to 1 digit).

Modified Newton's method

- simple root ($m = 1$):
  - quadratic convergence with Newton's method
- multiple root ($m \geq 2$):
  - linear convergence with Newton's method ☹
  - quadratic convergence with modified Newton's method ☺

Modified Newton's method

- simple root ($m = 1$):
  - quadratic convergence with Newton's method
- multiple root ($m \geq 2$):
  - linear convergence with Newton's method ☹
  - quadratic convergence with modified Newton's method ☺

    we compute the sequence $x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$

    m is required ☹

Modified Newton's method

- simple root ($m = 1$):
  - quadratic convergence with Newton's method
- multiple root ($m \geq 2$):
  - linear convergence with Newton's method ☹
  - quadratic convergence with modified Newton's method ☺

    we compute the sequence $x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$

    m is required ☹

How to compute m?

Estimation of the multiplicity m

Proposition [Yakoubsohn, 2003]

Let (x_n) be the sequence of approximations computed using Newton's method of the root α of multiplicity m of a polynomial. Then

$$\lim_{i \to \infty} \frac{x_{i+2} - x_{i+1}}{x_{i+1} - x_i} = 1 - \frac{1}{m}.$$

$\Rightarrow$ m can be estimated from 3 successive iterates of Newton's method

Modified Newton's method using SAM

Algorithm 1: Modified Newton with a requested accuracy using SAM

$step = 0$;
do
  $step = step + 1$;
  if $step = 1$ **then**
    $(x, m) =$ Newton(x_{init}) $\leftarrow$ optimal stopping criterion
  else
    x = Modified_Newton(x_{init},m) $\leftarrow$ optimal stopping criterion
  end
  $x_{init} = x$;
  double the working precision;
while $C_x \leq Requested\_accuracy$;

C_x: #correct digits of x estimated by SAM

Numerical experiments: modified Newton's method (1/3)

We compute using Algorithm 1 approximations of the root α of $P_m(x) = (3x-1)^m$ and for each approximation:

- #digits not affected by rounding errors estimated by SAM
- #digits in common with the exact root α: $\lfloor -\log_{10} \left| \frac{\alpha_{computed}-\alpha}{\alpha} \right| \rfloor$

m	requested accuracy	#digits SAM	#digits exact	time (s)
100	100	130	130	1.1E−1
	500	655	655	9.9E+0
	1000	1311	1311	6.6E+1
200	500	653	653	1.6E+1
	1000	1305	1305	1.2E+2
500	500	651	651	1.3E+1
	1000	1301	1301	2.7E+2

Numerical experiments: modified Newton's method (2/3)

We compute using Algorithm 1 approximations of the roots α_i of $P(x) = (19x+5)^5\,(19x+21)^9\,(19x+46)^{13}\,(19x+67)^{25}$ and for each approximation:

- #digits not affected by rounding errors estimated by SAM
- #digits in common with the exact root α_i: $\lfloor -\log_{10}\left|\frac{\alpha_{i\ computed}-\alpha_i}{\alpha_i}\right|\rfloor$

root	requested accuracy	#digits SAM	#digits exact	time (s)
$\alpha_1 = -5/19$	100	123	123	1.2E+0
	200	243	243	5.1E+0
	500	603	603	5.1E+1
$\alpha_2 = -21/19$	100	124	123	3.4E+0
	200	235	235	1.9E+1
	500	569	569	1.9E+2
$\alpha_3 = -46/19$	100	134	133	8.2E+0
	200	242	241	4.5E+1
	500	564	563	4.8E+2
$\alpha_4 = -67/19$	100	122	122	3.2E+1
	200	218	218	2.0E+2
	500	507	507	3.2E+3

Numerical experiments: modified Newton's method (3/3)

☺ For each root, the multiplicity is correctly determined.

☺ In the approximations provided by SAM, the digits not affected by rounding errors are always in common with the exact root, up to one.

☹ It may be difficult to estimate the required initial precision

- too low $\Rightarrow$ insignificant results
- too high $\Rightarrow$ costly computation
- depends on the requested accuracy and on the polynomial (multiplicity, number of roots,...)

Prospects:

- determine automatically the optimal initial precision
- theoretical results on the dynamical control of Newton's method $\rightarrow$ modified Newton's method

Conclusions/Perspectives

Conclusions:

- extension of CADNA to quadruple precision with a reasonable cost
- numerical validation of scientific codes in any working precision

Perspectives:

- numerical validation of parallel codes in quadruple precision (with OpenMP, MPI)
- with O3, *double-double* may perform better than *binary128*
  $\Rightarrow$ implementation of DSA based on *double-double*
  requires *double-double* algorithms with directed rounding
- precision optimization on FPGA using SAM

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 M.S. Ibrahim.
Dynamical control of Newton's method for multiple roots of polynomials.
Reliable Computing, 21, 2016.

[3] 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.

[4] 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

[5] S. Graillat, F. Jézéquel, S. Wang, and Y. Zhu.
Stochastic arithmetic in multiprecision.
Mathematics in Computer Science, 5(4):359–375, 2011.

[6] S. Graillat and F. Jézéquel.
Tight interval inclusions with compensated algorithms.
In *18th international symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2018)*, Tokyo, Japan, 2018.

[7] S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière.
PROMISE: floating-point precision tuning with stochastic arithmetic.
In *17th international symposium on Scientific Computing, Computer Arithmetic and Validated Numerics (SCAN 2016)*, Uppsala, Sweden, 2016.

[8] S. Graillat, F. Jézéquel, R. Picot, F. Févotte, and B. Lathuilière.
Numerical validation in quadruple precision using stochastic arithmetic.
In *TNC'18. Trusted Numerical Computations*, volume 8 of *Kalpa Publications in Computing*, pages 38–53, 2018.

References III

[9] 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.

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

[11] 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