Accurate polynomial evaluation in floating point arithmetic

Stef Graillat

Université de Perpignan Via Domitia
Laboratoire LP2A
Équipe de recherche en Informatique DALI

MIMS Seminar, February, 10^{th} 2006

General motivation

Provide numerical algorithms and software being

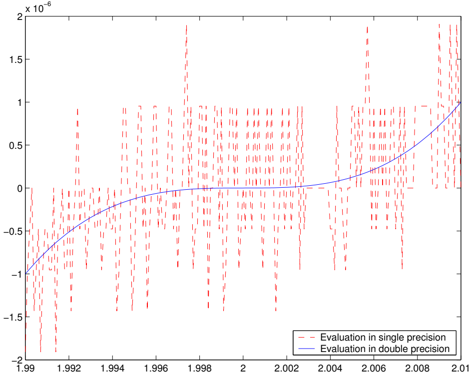
- a few times more accurate than the result from IEEE 754 working precision:
  ▷ the actual accuracy is proved to satisfy improved versions of the "classic rule of thumb";
- efficient in term of running-time without too much portability sacrifice:
  ▷ only working with IEEE 754 precision: single,double;
- together with a residual error bound to control the accuracy of the computed result:
  ▷ dynamic and validated error bound computable in IEEE 754 arithmetic.

Example for polynomial evaluation with Horner scheme:
→ the Compensated Horner Scheme[1]

[1] SG, N. Louvet, Ph. Langlois. Compensated Horner Scheme. Submitted to SISC

Loss of accuracy in the polynomial evaluation

Evaluation of the polynomial $p(x) = (x-2)^3 = x^3 - 6x^2 + 12x - 8$ for about 200 points near $x = 2$ in single and double precision



Problems in finite precision computation

Aims : Solving the previous problems being accurate and reliable

- Understanding the influence of the finite precision on the numerical quality of numerical software
  - inaccurate results;
  - numerical instabilities.
- controlling and limiting harmful effect

How to be more accurate without large overheads?

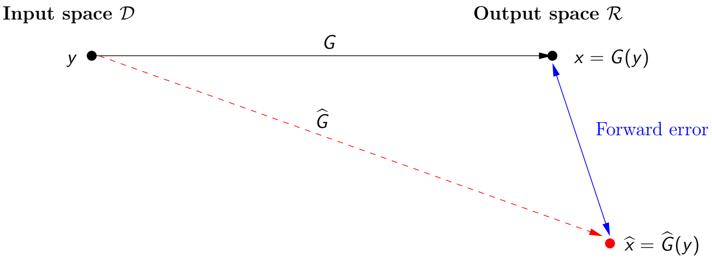
Problems in computing with uncertainties

Understanding the difficulties to deal with uncertainties:

- Controlling the effects of uncertainties:
  - How to measure the difficulty of solving the problem?
  - How to appreciate the reliability of the algorithm?
  - How to estimate the accuracy of the computed solution?
- Limiting the effect of finite precision
  - How to improve the accuracy of the solution?

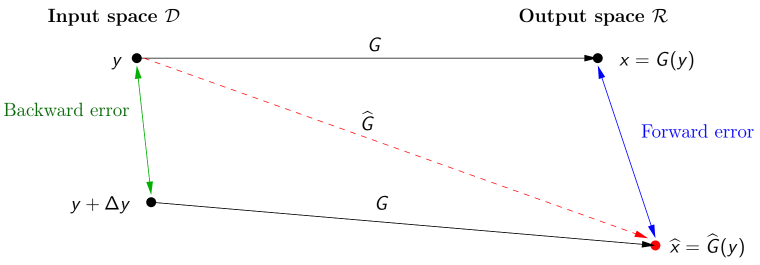
Which notions to answer these questions?

Error analysis



- Forward error analysis
- Backward error analysis
  Identify $\widehat{x}$ as the solution of a perturbed problem:
  $\widehat{x} = G(y + \Delta y)$.

Error analysis



- Forward error analysis
- Backward error analysis
  Identify $\widehat{x}$ as the solution of a perturbed problem:
  $\widehat{x} = G(y + \Delta y)$.

Advantages of backward error analysis

- **How to estimate the accuracy of the computed solution?**
  At the first order, we have the rule of thumb:

  forward error $\lesssim$ condition number $\times$ backward error.

- **How to measure the difficulty of solving the problem ?**
  Condition number measures the sensitivity of the solution to perturbation in the data

  Condition number : $K(P,y) := \lim_{\varepsilon \to 0} \sup_{\Delta y \in \mathcal{P}(\varepsilon)} \left\{ \frac{\|\Delta x\|_{\mathcal{R}}}{\|\Delta y\|_{\mathcal{D}}} \right\}$

- **How to appreciate the reliability of the algorithm?**
  Backward error measures the distance between the problem we solved and the initial problem.

  Backward error : $\eta(\widehat{x}) = \min_{\Delta y \in \mathcal{D}} \{ \|\Delta y\|_{\mathcal{D}} : \widehat{x} = G(y + \Delta y) \}$

Advantages of backward error analysis

- **How to estimate the accuracy of the computed solution?**
  At the first order, we have the rule of thumb:

  forward error $\lesssim$ condition number $\times$ backward error.

- **How to measure the difficulty of solving the problem ?**
  Condition number measures the sensitivity of the solution to perturbation in the data

  Condition number : $K(P, y) := \lim_{\varepsilon \to 0} \sup_{\Delta y \in \mathcal{P}(\varepsilon)} \left\{ \frac{\|\Delta x\|_{\mathcal{R}}}{\|\Delta y\|_{\mathcal{D}}} \right\}$

- **How to appreciate the reliability of the algorithm?**
  Backward error measures the distance between the problem we solved and the initial problem.

  Backward error : $\eta(\widehat{x}) = \min_{\Delta y \in \mathcal{D}} \{\|\Delta y\|_{\mathcal{D}} : \widehat{x} = G(y + \Delta y)\}$

Advantages of backward error analysis

- **How to estimate the accuracy of the computed solution?**
  At the first order, we have the rule of thumb:

  forward error $\lesssim$ condition number $\times$ backward error.

- **How to measure the difficulty of solving the problem ?**
  Condition number measures the sensitivity of the solution to perturbation in the data

  Condition number : $K(P, y) := \lim_{\varepsilon \to 0} \sup_{\Delta y \in \mathcal{P}(\varepsilon)} \left\{ \frac{\|\Delta x\|_{\mathcal{R}}}{\|\Delta y\|_{\mathcal{D}}} \right\}$

- **How to appreciate the reliability of the algorithm?**
  Backward error measures the distance between the problem we solved and the initial problem.

  Backward error : $\eta(\widehat{x}) = \min_{\Delta y \in \mathcal{D}} \{\|\Delta y\|_{\mathcal{D}} : \widehat{x} = G(y + \Delta y)\}$

Outline

1. Motivations
2. Accurate polynomial evaluation

Floating point number

Floating point system $\mathbb{F} \subset \mathbb{R}$:

$$x = \pm \underbrace{x_0.x_1 \ldots x_{p-1}}_{mantissa} \times b^e, \quad 0 \le x_i \le b-1, \quad x_0 \neq 0$$

b : basis, p : precision, e : exponent range s.t. $e_{\min} \le e \le e_{\max}$

Machine epsilon $\epsilon = b^{1-p}$, $|1^+ - 1| = \epsilon$

Approximation of $\mathbb{R}$ by $\mathbb{F}$, rounding $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$
Let $x \in \mathbb{R}$ then

$$\mathrm{fl}(x) = x(1+\delta), \quad |\delta| \le \mathbf{u}.$$

Unit roundoff $\mathbf{u} = \epsilon/2$ for round-to-nearest

Standard model of floating point arithmetic

Let $x, y \in \mathbb{F}$,

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \quad |\delta| \le \mathbf{u}, \quad \circ \in \{+, -, \cdot, /\}$$

IEEE 754 standard (1985)

Type	Size	Mantissa	Exponent	Unit roundoff	Range
Single	32 bits	23+1 bits	8 bits	$\mathbf{u} = 2^{-24} \approx 5,96 \times 10^{-8}$	$\approx 10^{\pm 38}$
Double	64 bits	52+1 bits	11 bits	$\mathbf{u} = 2^{-53} \approx 1,11 \times 10^{-16}$	$\approx 10^{\pm 308}$

For a more precise evaluation scheme

- Accurate evaluation of $p(x)$: the compensated Horner scheme and the compensated rule of thumb
- An improved and validated error bound
- Theoretical and experimental results exhibit the
  - actual accuracy: twice the current working precision behavior,
  - actual speed: twice faster than the corresponding double-double implementation

More accuracy, how ?

More internal precision:

- hardware
  - extended precision in x86 architecture
- software
  - fixed length expansions libraries: double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
  - arbitrary length expansions libraries: Priest, Shewchuk
  - arbitrary multiprecision libraries: MP, MPFUN/ARPREC, MPFR

Correcting rounding errors:

- compensated summation (Kahan,1965) and doubly compensated summation (Priest,1991), etc.
- accurate sum and dot product: Ogita, Rump and Oishi (2005) $\longrightarrow$ twice the current working precision behavior and fast compared to double-double library

More accuracy, how ?

More internal precision:

- hardware
  - extended precision in x86 architecture
- software
  - fixed length expansions libraries: double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
  - arbitrary length expansions libraries: Priest, Shewchuk
  - arbitrary multiprecision libraries: MP, MPFUN/ARPREC, MPFR

Correcting rounding errors:

- compensated summation (Kahan,1965) and doubly compensated summation (Priest,1991), etc.
- accurate sum and dot product: Ogita, Rump and Oishi (2005) $\longrightarrow$ twice the current working precision behavior and fast compared to double-double library

At current working precision ...

Rule of thumb for backward stable algorithms :

solution accuracy $\approx$ condition number $\times$ computing precision

1. IEEE-754 precision: double ($\mathbf{u} = 2^{-53} \approx 10^{-16}$)
2. Condition number for the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$:

$$\text{cond}(p,x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\widetilde{p}(|x|)}{|p(x)|}, \text{ always } \geq 1.$$

3. Accuracy of the solution $\widehat{p}(x)$:

$$\frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \leq \alpha(n) \times \text{cond}(p,x) \times \mathbf{u}$$

with $\alpha(n) \approx 2n$

What means "twice the working precision behavior"?

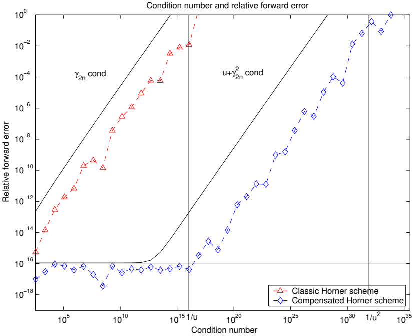
Compensated rule of thumb:

solution accuracy $\lesssim$ precision + condition number $\times$ precision^2

Three regimes in precision for the evaluation of $\widehat{p}(x)$:

1) condition number $\leq 1/\mathbf{u}$: the accuracy of $\widehat{p}(x)$ is optimal

$$\frac{|\widehat{p}(x) - p(x)|}{|p(x)|} \approx \mathbf{u}$$



What means "twice the working precision behavior"?

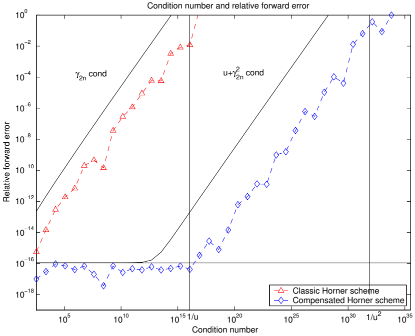
Compensated rule of thumb:

solution accuracy $\lesssim$ precision + condition number $\times$ precision^2

Three regimes in precision for the evaluation of $\widehat{p}(x)$:

2) $1/\mathbf{u} \leq$ condition number$\leq 1/\mathbf{u}^2$: the result $\widehat{p}(x)$ verifies

$$\frac{|\widehat{p}(x) - p(x)|}{|p(x)|} \approx \text{cond} \times \mathbf{u}^2$$



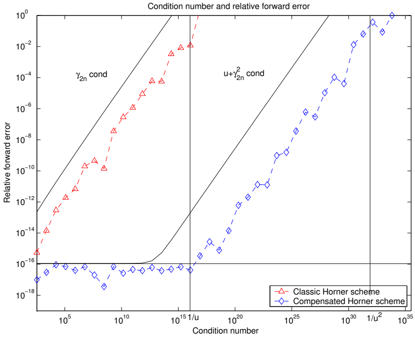
What means "twice the working precision behavior"?

Compensated rule of thumb:

$$\text{solution accuracy} \lesssim \text{precision} + \text{condition number} \times \text{precision}^2$$

Three regimes in precision for the evaluation of $\widehat{p}(x)$:

3) no more accuracy when condition number $> 1/\mathbf{u}^2$.



The Horner scheme

Algorithm 1 (Horner scheme)

function $\texttt{res} = \texttt{Horner}(p, x)$
  $s_n = a_n$
  for $i = n - 1 : -1 : 0$
    $p_i = \mathrm{fl}(s_{i+1} \cdot x)$ % rounding error π_i
    $s_i = \mathrm{fl}(p_i + a_i)$ % rounding error σ_i
  end
  $\texttt{res} = s_0$

$\gamma_n = n\mathbf{u}/(1 - n\mathbf{u}) \approx n\mathbf{u}$

$$\frac{|p(x) - \texttt{Horner}(p, x)|}{|p(x)|} \leq \underbrace{\gamma_{2n}}_{\approx 2n\mathbf{u}} \mathrm{cond}(p, x)$$

Error-free transformations for sum

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

For the sum, algorithms by Dekker (1971) and Knuth (1974)

Algorithm 2 (Error-free transformation of the sum of 2 floating point numbers, needs $|a| \geq |b|$)

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

Algorithm 3 (Error-free transformation of the sum of 2 floating point numbers)

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

Error-free transformations for product (1/3)

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

For the product : algorithm `TwoProduct` by Veltkamp and Dekker (1971)

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

Algorithm 4 (Error-free split of a floating point number into two parts)

function $[x, y] = \mathtt{Split}(a, b)$
  $\mathtt{factor} = 2^s + 1$
  $c = \mathrm{fl}(\mathtt{factor} \cdot a)$
  $x = \mathrm{fl}(c - (c - a))$
  $y = \mathrm{fl}(a - x)$

Error-free transformations for product (2/3)

Algorithm 5 (Error-free transformation of the product of two floating point numbers)

function $[x, y] = \mathtt{TwoProduct}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $[a_1, a_2] = \mathtt{Split}(a)$
  $[b_1, b_2] = \mathtt{Split}(b)$
  $y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

Error-free transformations for product (3/3)

What is a Fused Multiply and Add (FMA) in floating point arithmetic?
$\longrightarrow$ Given a, b and c three floating point numbers, $\texttt{FMA}(a, b, c)$ computes $a \cdot b + c$ rounded according to the current rounding mode
$\Rightarrow$ only one rounding error for two operations!
FMA is available on Intel Itanium, IBM RS/6000, IBM Power PC, etc.

Algorithm 6 (Error-free transformation of the product of two floating point numbers with FMA)

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

Error-free transformation for the Horner scheme

$$\boxed{p(x) = \texttt{Horner}(p, x) + (p_\pi + p_\sigma)(x)}$$

Algorithm 7 (Error-free transformation for the Horner scheme)

function $[\texttt{Horner}(p, x), p_\pi, p_\sigma] = \texttt{EFTHorner}(p, x)$
$s_n = a_n$
for $i = n - 1 : -1 : 0$
  $[p_i, \pi_i] = \texttt{TwoProduct}(s_{i+1}, x)$
  $[s_i, \sigma_i] = \texttt{TwoSum}(p_i, a_i)$
  Let π_i be the coefficient of degree i of p_π
  Let σ_i be the coefficient of degree i of p_σ
end
$\texttt{Horner}(p, x) = s_0$

Compensated Horner scheme

Algorithm 8 (Compensated Horner scheme)

function $\texttt{res} = \texttt{CompHorner}(p, x)$
$[h, p_\pi, p_\sigma] = \texttt{EFTHorner}(p, x)$
$c = \texttt{Horner}(p_\pi + p_\sigma, x)$
$\texttt{res} = \mathrm{fl}(h + c)$

Accuracy of the compensated Horner scheme

Theorem 1

Let p be a polynomial of degree n with floating point coefficients, and x be a floating point value. Then if no underflow occurs,

$$\frac{|\texttt{CompHorner}(p,x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \underbrace{\gamma_{2n}^2}_{\approx 4n^2\mathbf{u}^2} \operatorname{cond}(p,x).$$

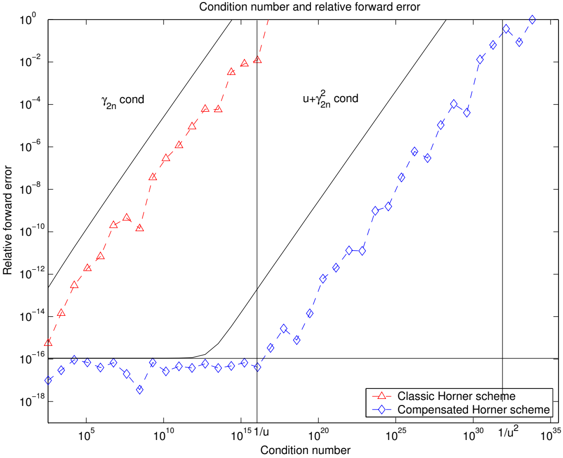
- Key point in the proof:

$$(\widetilde{p_\pi} + \widetilde{p_\sigma})(|x|) \leq \gamma_{2n}\widetilde{p}(|x|)$$

- a similar bound is proved in presence of underflow

Numerical experiments: testing the accuracy

Evaluation of $p_n(x) = (x-1)^n$ for $x = \mathrm{fl}(1.333)$ and $n = 3, \ldots, 42$



Numerical experiments: testing the speed efficiency

We compare

- `Horner`: IEEE 754 double precision Horner scheme
- `CompHorner`: our Compensated Horner scheme
- `DDHorner`: Horner scheme with internal double-double computation

All computations are performed in C language and IEEE 754 double precision

For every polynomials p_n with n varying from 3 to 42:

- we perform 100 runs measuring (100) numbers of cycles (TSC counter for IA-32),
- we keep the mean value, the min and the max of the 10 smallest numbers of cycles.

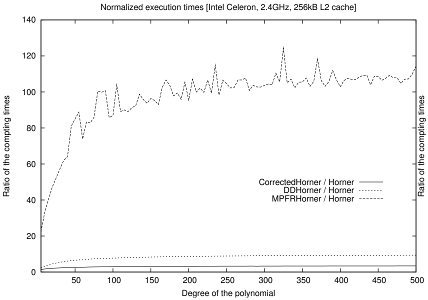
Speed efficiency: measured and theoretical ratios

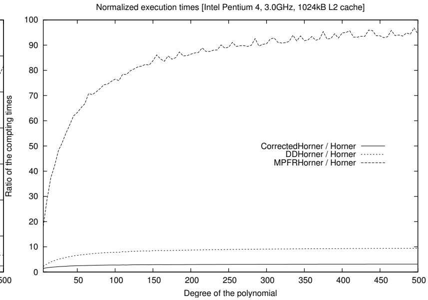
Pentium 4: 3.0GHz, 1024kB cache L2 - GCC 3.4.1				
ratio	minimum	mean	maximum	theoretical
CompHorner/Horner	1.5	2.9	3.2	13
DDHorner/Horner	2.3	8.4	9.4	17

Intel Celeron: 2.4GHz, 256kB cache L2 - GCC 3.4.1				
ratio	minimum	mean	maximum	theoretical
CompHorner/Horner	1.4	3.1	3.4	13
DDHorner/Horner	2.3	8.4	9.4	17

$\longrightarrow$ compensated Horner scheme = Horner scheme with double-double without renormalization

The corrected algorithm runs twice faster than corresponding double-double





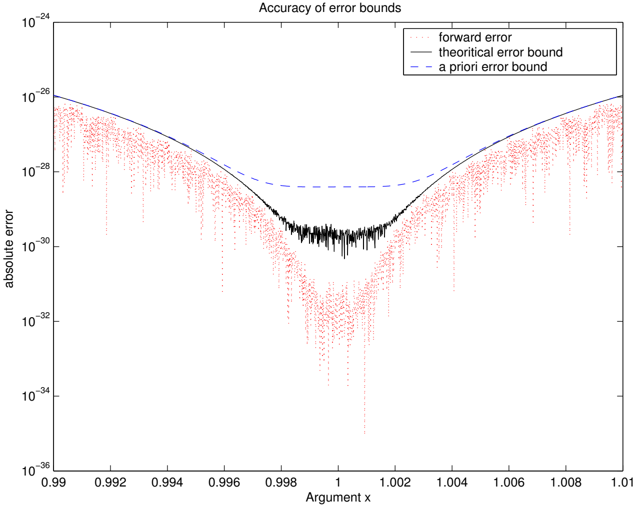
A dynamic error bound

Theorem 2

Given a polynomial p of degree n with floating point coefficients, and a floating point value x, we consider $\texttt{res} = \texttt{CompHorner}(p, x)$*. The absolute forward error affecting the evaluation is bounded according to*

$$|\texttt{CompHorner}(p,x) - p(x)| \leq \mathrm{fl}((\mathbf{u}|\texttt{res}| + (\gamma_{4n+2}\texttt{Horner}(\widetilde{p_\pi} + \widetilde{p_\sigma}, |x|) + 2\mathbf{u}^2|\texttt{res}|))).$$

Accuracy of the bound for $p_5(x) = (x-1)^5$



Conclusion and future work

- The compensated Horner scheme provides
  - actual accuracy as doubling the working precision,
  - actual speed being twice faster than the corresponding double-double subroutine,
  - together with a dynamic and validated error bound.
- Past, current and future developments
  - Compensated Horner scheme: underflow, with `FMA`, for `FMA`
  - same techniques with Newton methods

The new revision of IEEE 754 standard should include `tailadd`, `tailsubtract` and `tailmultiply` that compute the error during an addition, a subtraction and a multiplication.

Thank you for your attention