Floating-point arithmetic and error analysis (AFAE)

# Increasing the accuracy, examples with polynomials

**Stef Graillat**

LIP6/PEQUAN - Sorbonne University

Lecture Master 2 SFPN - MAIN5

# Outline

# Outline

# Understanding the difficulties when computing with finite precision

Controlling the effects of finite precision:

- How to measure the difficulty of solving the problem?
- How to characterize the reliability of the algorithm?
- How to estimate the accuracy of the computed solution?

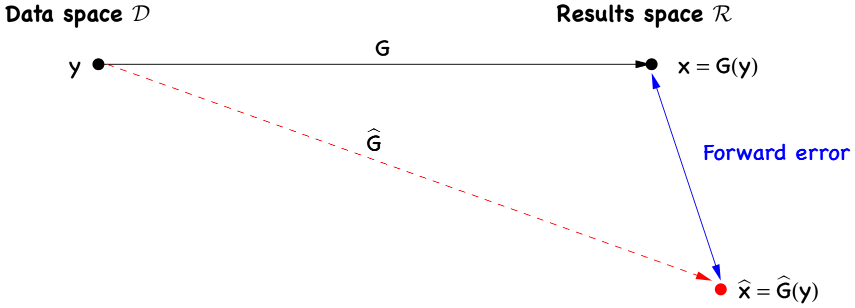
Limiting the effects of finite precision

- How to improve the accuracy of the solution?

How to answer these questions?

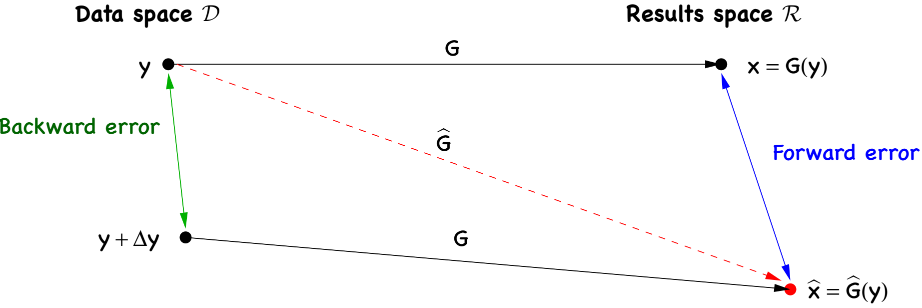
# Outline

# Error analysis (Wilkinson, Higham)




Forward error analysis

# Error analysis (Wilkinson, Higham)




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

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

# Advantages of backward error analysis

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

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

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

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

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

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

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

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

# Advantages of backward error analysis

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

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

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

Backward error : $\eta(\widehat{x}) = \mathbf{u} \longrightarrow$ backward stable

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

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

# Advantages of backward error analysis

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

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

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

Backward error : $\eta(\widehat{x}) = \mathbf{u} \longrightarrow$ backward stable

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

forward error $\lesssim$ condition number $\times$ **u**.

# Achieving more accuracy with compensated algorithms

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
- arbitrary multiprecision libraries: MP, MPFUN/ARPREC, MPFR
- compensated algorithms (e.g. Kahan, Priest, Ogita-Rump-Oishi)

Error-free transformations (EFT) (Dekker, Knuth) are properties and algorithms to compute the elementary rounding errors,

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

# EFT for the summation

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

Algorithms of Dekker (1971) and Knuth (1974)

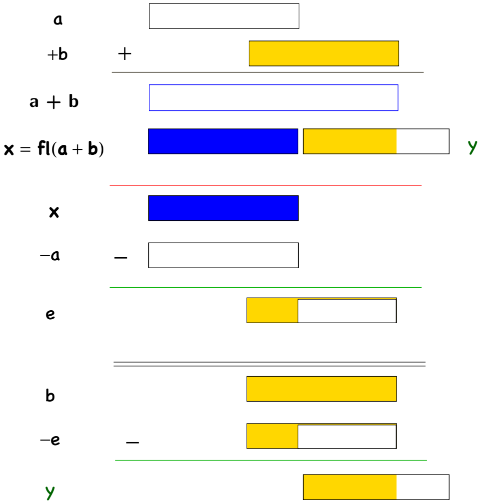
**Algorithm (EFT of the sum of 2 floating-point numbers with $|a| \geq |b|$)**

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

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

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

# EFT for the summation



# Error bound for EFT of the sum

**Theorem**

Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] = \mathrm{TwoSum}(a, b)$. Then,

$$a + b = x + y, \quad x = a \oplus b, \quad |y| \leq u|x|, \quad |y| \leq u|a + b|.$$

The algorithm `TwoSum` requires 6 flops.

# EFT for the product (1/3)

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

Algorithm `TwoProduct` by Veltkamp and Dekker (1971)

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

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

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

# EFT for the product (2/3)

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

function $[x, y]$ = $\texttt{TwoProduct}(a, b)$
  $x = a \otimes b$
  $[a_1, a_2]$ = $\texttt{Split}(a)$
  $[b_1, b_2]$ = $\texttt{Split}(b)$
  $y$ = $a_2 \otimes b_2 \ominus (((x \ominus a_1 \otimes b_1) \ominus a_2 \otimes b_1) \ominus a_1 \otimes b_2)$

**Theorem**

Let $a, b \in \mathbb{F}$ and let $x, y \in \mathbb{F}$ such that $[x, y] = \texttt{TwoProduct}(a, b)$ . Then,

$$a \cdot b = x + y, \quad x = a \otimes b, \quad |y| \leq u|x|, \quad |y| \leq u|a \cdot b|,$$

The algorithm $\texttt{TwoProduct}$ requires 17 flops.

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

$\mathrm{FMA}(a, b, c)$ is the nearest floating-point number $a \cdot b + c \in \mathbb{F}$

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

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

The FMA is available for example on PowerPC, Itanium, Cell, Xeon Phi, Haswell processors.

# Outline

# Recursive summmation algorithm

Computation of $s = \sum_{i=1}^{n} p_i$

## Algorithm (Classic summation algorithm)

**function** res = Sum($p$)

$\sigma = 0$;

**for** $i = 1 : n$

$\sigma = \sigma \oplus p_i$

res $= \sigma$

# Rounding error analysis(1/2)

## Lemma 1

If $|\delta_i| \leq \mathbf{u}$, $\rho_i = \pm 1$ for $i = 1 : n$ and $nu < 1$ then

$$\prod_{i=1}^{n} (1 + \delta_i)^{\rho_i} = 1 + \delta_n,$$

where

$$|\delta_n| \leq \frac{nu}{1 - nu} =: \gamma_n.$$

# Rounding error analysis (2/2)

**Theorem**

With the previous notations, we have

$$|\mathtt{res} - s| \le \gamma_{n-1} \sum_{i=1}^{n} |p_i|.$$

# Kahan's compensated summation algorithm

**Algorithm (Kahan's algorithm)**

**function** $\texttt{res} = \texttt{SCompSum}(p)$
  $\sigma = 0$
  $e = 0$
  **for** $i = 1 : n$
    $y = p_i \oplus e$
    $[\sigma, e] = \texttt{FastTwoSum}(\sigma, y)$
  $\texttt{res} = \sigma$

# Rounding error analysis

## Theorem

With the previous notations, we have

$$|\mathrm{res} - s| \leq (2u + \mathcal{O}(nu^2)) \sum_{i=1}^{n} |p_i|.$$

# Priest's doubly compensated summation algorithm

## Algorithm (Priest's algorithm)

function $\texttt{res}$ = $\mathrm{DCompSum}(p)$

  sort the $p_i$ such that $|p_1| \geq |p_2| \geq \cdots \geq |p_n|$

  $s = 0$

  $c = 0$

  for $i = 1 : n$

    $[y, u] = \mathtt{FastTwoSum}(c, p_i)$

    $[t, v] = \mathtt{FastTwoSum}(y, s)$

    $z = u \oplus v$

    $[s, c] = \mathtt{FastTwoSum}(t, z)$

  $\mathtt{res} = s$

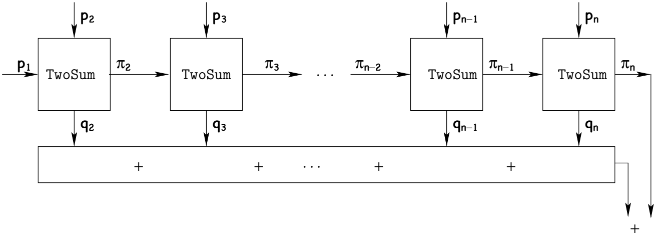
# Rounding error analysis

**Theorem**

With the previous notations, we have

$$|\mathtt{res} - \mathbf{s}| \leq \mathbf{2u}|\mathbf{s}|$$

# Compensated algorithm of Ogita, Rump and Oishi



# Compensated algorithm of Ogita, Rump and Oishi

## Algorithm (Compensated algorithm)

**function** res = CompSum($p$)

$\pi_1 = p_1$ ; $\sigma_1 = 0$;

**for** $i = 2 : n$

$[\pi_i, q_i] = \mathrm{TwoSum}(\pi_{i-1}, p_i)$

$\sigma_i = \sigma_{i-1} \oplus q_i$

$\mathrm{res} = \pi_n \oplus \sigma_n$

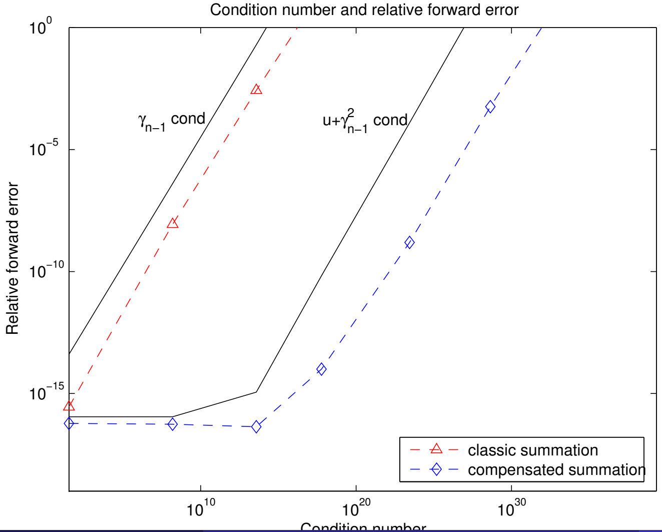
# Compensated algorithm of Ogita, Rump and Oishi

## Proposition

Let us apply CompSum Algorithm to $p_i \in \mathbb{F}$, $1 \leq i \leq n$. Let $s := \sum p_i$, $S := \sum |p_i|$ and $nu < 1$. Then, we have

$$|\mathrm{res} - s| \leq u|s| + \gamma_{n-1}^2 S. \qquad (1)$$

# Compensated algorithm of Ogita, Rump and Oishi



# Outline

# Compensated dot product (1/2)

**Algorithm (Compensated dot product algorithm)**

**function** $\texttt{res} = \texttt{CompDot}(x, y)$
  **for** $i = 1 : n$
    $[r_i, r_{n+i}] = \texttt{TwoProduct}(x_i, y_i)$
  $\texttt{res} = \texttt{CompSum}(r)$

# Compensated dot product (2/2)

## Algorithm (Compensated dot product algorithm)

function $\mathtt{res} = \mathtt{CompDot2}(x, y)$
  $[p, s] = \mathtt{TwoProduct}(x_1, y_1)$
  for $i = 2 : n$
    $[h, r] = \mathtt{TwoProduct}(x_i, y_i)$
    $[p, q] = \mathtt{TwoSum}(p, h)$
    $s = s \oplus (q \oplus r)$
  end
  $\mathtt{res} = p \oplus s$

## Proposition

Let floating point numbers $x_i, y_i \in \mathbb{F}$, $1 \leq i \leq n$, be given and denote by $\mathtt{res} \in \mathbb{F}$ the result computed by Algorithm $\mathtt{CompDot2}$. Then

$$|\mathtt{res} - x^T y| \leq \mathbf{u}|x^T y| + \gamma_n^2 |x^T||y|.$$

# Outline

# Horner scheme

## Algorithm

function $\mathtt{res} = \mathtt{Horner}(p, x)$ % $p(x) = \sum_{i=0}^{n} a_i x^i$

$s_n = a_n$

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

$p_i = s_{i+1} \otimes x$

$s_i = p_i \oplus a_i$

end

$\mathtt{res} = s_0$

Condition number for the evaluation of $p(x)$:

$$\mathrm{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\widetilde{p}(|x|)}{|p(x)|}$$

Relative error bound: $$\frac{|p(x) - \mathtt{Horner}(p, x)|}{|p(x)|} \leq \underbrace{\gamma_{2n}}_{\approx 2nu} \mathrm{cond}(p, x)$$

# Horner scheme

## Algorithm

function $\texttt{res} = \texttt{Horner}(p, x)$ % $p(x) = \sum_{i=0}^{n} a_i x^i$

$s_n = a_n$

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

$p_i = s_{i+1} \otimes x$ % rounding error $\pi_i$

$s_i = p_i \oplus a_i$ % rounding error $\sigma_i$

end

$\texttt{res} = s_0$

Condition number for the evaluation of $p(x)$:

$$\mathrm{cond}(p, x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\widetilde{p}(|x|)}{|p(x)|}$$

Relative error bound: $$\frac{|p(x) - \texttt{Horner}(p, x)|}{|p(x)|} \leq \underbrace{\gamma_{2n}}_{\approx 2nu} \mathrm{cond}(p, x)$$

# EFT for Horner scheme

## Algorithm

**function** $[\mathrm{Horner}(p,x), p_\pi, p_\sigma] = \mathrm{EFTHorner}(p,x)$

$s_n = a_n$

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

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

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

**end**

$\mathrm{Horner}(p,x) = s_0$

$$p_\pi(x) = \sum_{i=0}^{n-1} \pi_i x^i, \qquad p_\sigma(x) = \sum_{i=0}^{n-1} \sigma_i x^i$$

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

# Compensated Horner scheme (CHS) and its accuracy

## Algorithm (CHS)

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

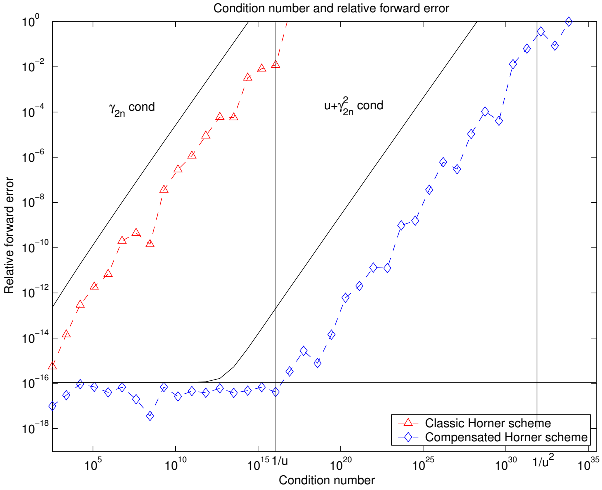
## Theorem

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

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

# Numerical experiments: testing the accuracy

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



# Numerical experiments: testing the speed efficiency

We compare

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

All computations are performed in C and IEEE 754 double precision

| ratio | minimum | mean | maximum | theoretical |
|---|---|---|---|---|
| `CompHorner`/`Horner` | 1.5 | 2.9 | 3.2 | 13 |
| `DDHorner`/`Horner` | 2.3 | 8.4 | 9.4 | 17 |

# Compensated Horner Derivative algorithm

The Horner Derivative (HD) algorithm is the classic method for the evaluation of the k-derivative of a polynomial p(x)

## Algorithm (HD)

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

$y_i^j = 0$ for $i = 0 : 1 : k$ and $j = n + 1 : -1 : 0$

$y_{-1}^{j+1} = a_j$ for $j = n : -1 : 0$

for $j = n : -1 : 0$

  for $i = \min(k, n - j) : -1 : \max(0, k - j)$

    $y_i^j = x \otimes y_i^{j+1} \oplus y_{i-1}^{j+1}$

  end

end

$\texttt{res} = k! \otimes y_k^0$

## Algorithm (CHD)

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

$y_i^j = 0$, $\widehat{\varepsilon y}_i^j = 0$, for $i = 0 : 1 : k$, and $j = n + 1 : -1 : 0$

$y_{-1}^{j+1} = a_j$, $\widehat{\varepsilon y}_{-1}^{j+1} = 0$, for $j = n : -1 : 0$

for $j = n : -1 : 0$

  for $i = \min(k, n - j) : -1 : \max(0, k - j)$

    $[s, \pi_i^j] = \texttt{TwoProd}(x, \widehat{y}_i^{j+1})$

    $[\widehat{y}_i^j, \sigma_i^j] = \texttt{TwoSum}(s, \widehat{y}_{i-1}^{j+1})$

    $\widehat{\varepsilon y}_i^j = x \otimes \widehat{\varepsilon y}_i^{j+1} \oplus \widehat{\varepsilon y}_{i-1}^{j+1} \oplus (\pi_i^j \oplus \sigma_i^j)$

  end

end

$\texttt{res} = (\widehat{y}_k^0 \oplus \widehat{\varepsilon y}_k^0) \otimes k!$

# Rounding error analysis of CHD algorithm

## Theorem

Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating-point coefficients, and x a floating-point value (with $p^{(k)}(x) \neq 0$). The relative forward error bound in CHD algorithm is such that

$$\frac{|\mathrm{CompHD}(p, x, k) - p^{(k)}(x)|}{|p^{(k)}(x)|} \leq 2u + (k+1) \underbrace{\gamma_{2n}\gamma_{3n}}_{\approx 6n^2u^2} \mathrm{cond}(p, x, k).$$

The condition number for the k-th derivative evaluation of a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ at entry x is given by

$$\mathrm{cond}(p, x, k) = \frac{k! \sum_{m=k}^{n} \binom{m}{k} |x|^{m-k} |a_m|}{|k! \sum_{m=k}^{n} \binom{m}{k} x^{m-k} a_m|} = \frac{\widetilde{p}^{(k)}(x)}{|p^{(k)}(x)|},$$

# Numerical experiments

Average ratios of the floating-point operations

| $\frac{\text{CompHD}}{\text{HD}}$ | $\frac{\text{DDHD}}{\text{HD}}$ | $\frac{\text{CompHD}}{\text{DDHD}}$ |
|---|---|---|
| 8.35 | 13.60 | 61% |

Measured running time ratios

| | $\frac{\text{CompHD}}{\text{HD}}$ | $\frac{\text{DDHD}}{\text{HD}}$ | $\frac{\text{CompHD}}{\text{DDHD}}$ |
|---|---|---|---|
| Linux gcc 4.4.5 | 3.85 | 8.14 | 47% |
| Windows Vc++9.0 | 4.58 | 9.79 | 47% |

# Condition number for root finding

## Definition

Let $p(z) = \sum_{i=0}^{n} a_i z^i$ be a polynomial of degree n and x be a simple zero of p. The condition number of x is defined by

$$\mathrm{cond}_{\mathrm{root}}(p, x) = \lim_{\varepsilon \to 0} \sup \left\{ \frac{|\Delta x|}{\varepsilon |x|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}.$$

## Theorem

Let $p(z) = \sum_{i=0}^{n} a_i z^i$ be a polynomial of degree n and x be a simple zero of p. The condition number of x is given by

$$\mathrm{cond}_{\mathrm{root}}(p, x) = \frac{\widetilde{p}(|x|)}{|x||p'(x)|},$$

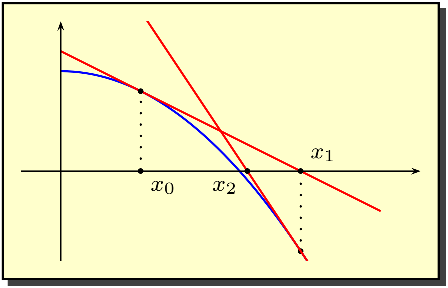
with $\widetilde{p}(x) = \sum_{i=0}^{n} |a_i| z^i$.

# Application to Newton's method

## Algorithm (The classic Newton's method)

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\mathrm{Horner}(p, x_i)}{\mathrm{HD}(p, x_i, 1)}$$

$$\frac{|x_{i+1} - x|}{|x|} \approx \gamma_{2n}\, \mathrm{cond}_{\mathrm{root}}(p, x) \qquad \text{[Higham, 1996]}$$



# Application to Newton's method

## Algorithm (The accurate Newton's method)

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\texttt{CompHorner}(p,x_i)}{\texttt{HD}(p,x_i,1)}$$

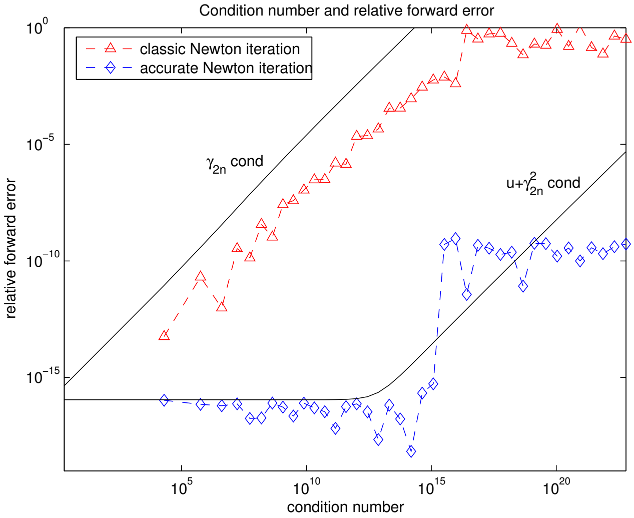
## Theorem

Assume that there is an $x$ such that $p(x) = 0$ and $p'(x) \neq 0$ is not too small. Assume also that $\mathbf{u} \cdot \mathrm{cond}_{\mathtt{root}}(p,x) \leq 1/8$. Then, for all $x_0$ such that $\beta|p'(x)^{-1}||x_0 - x| \leq 1/8$, Newton's method in floating-point arithmetic generates a sequence of $\{x_i\}$ whose relative error decreases until the first $i$ for which

$$\frac{|x_{i+1} - x|}{|x|} \approx \mathbf{u} + \gamma_{2n}^2 \,\mathrm{cond}_{\mathtt{root}}(p,x).$$

# Application to Newton's method

Test with $p_n(x) = (x-1)^n - 10^{-8}$ and $x = 1 + 10^{-8/n}$ for $n = 1 : 40$
$\text{cond}(p_n, x)$ varies from $10^4$ to $10^{22}$



# Application to Newton's method

**Algorithm (The new accurate Newton's method)**

$$x_0 = \xi$$
$$x_{i+1} = x_i - \frac{\texttt{CompHorner}(p, x_i)}{\texttt{CompHD}(p, x_i, 1)}$$

It is proved that

that the convergence of iterations strongly depends on the accuracy of the derivative's evaluation when the problem of finding simple root is too ill-conditioned, and

that the accuracy of the final iteration result depends on the accuracy with which the residual is computed.

# Application to Newton's method

It is shown that

In case of classic Newton's algorithm:

$$\left|\frac{x_i - x}{x}\right| < C\gamma_{2n}\,\mathrm{cond}_{\mathrm{root}}(p, x).$$

In case of accurate Newton's algorithms:

$$\left|\frac{x_i - x}{x}\right| < Ku + D\gamma_{2n}^2\,\mathrm{cond}_{\mathrm{root}}(p, x).$$

where C, K and D are small factors.

# Application to Newton's method

Assume that the simple root is $\alpha$ such that $f(\alpha) = 0$, $f'(\alpha) \neq 0$ with f is continuously differentiable in a neighborhood of the root, and in floating point arithmetic the computation of the derivative satisfies

Assumption 1 :
$$\left|\frac{\widehat{f'}(v) - f'(v)}{f'(v)}\right| < \omega < \frac{1}{2},$$

Assume also that for any v, obtained from the iteration from the initial value $v_0$ sufficiently close to the root $\alpha$, satisfies

Assumption 2 :
$$0 < \frac{f(v)}{f'(v)(v-\alpha)} < \mu_1.$$

In the iterative process, $f'(v) \neq 0$ and $\widehat{f'}(v) \neq 0$, meanwhile $\omega$ and $\mu_1$ satisfy

Assumption 3 :
$$\mu_1 + 2\omega \leq 2.$$

# Application to Newton's method

Newton's method or its improved versions in floating point arithmetic generates a sequence $\{\widehat{v}_i\}$ converging to $v_*$. Then assume that, when the iteration converges, there is

Assumption 4 : $$0 < \mu_2 < \frac{f(v_*)}{f'(v_*)(v_* - \alpha)}.$$

The parameters $\omega, \mu_1$ and $\mu_2$ used in Assumption 1-4 will help to obtain the accuracies guaranteed by the algorithms as follows.
In case of classic Newton's algorithm:

$$\left|\frac{\alpha - v_*}{v_*}\right| < C\gamma_{2n}\mathrm{cond}_{\mathrm{root}}(p, v_*).$$

# Application to Newton's method

In case of accurate Newton's algorithms:

$$\left|\frac{\alpha - \mathbf{v}_*}{\mathbf{v}_*}\right| < \mathbf{K}\mathbf{u} + \mathbf{D}\gamma_{\mathbf{2n}}^{\mathbf{2}}\mathrm{cond}_{\mathrm{root}}(\mathbf{p}, \mathbf{v}_*).$$

where C, K and D are the constants consist of $\omega$ and $\mu_2$.

# Numerical experiments

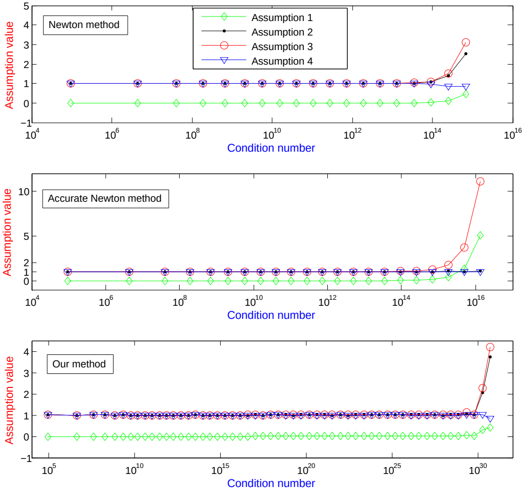
Computing the simple real zero of the expanded form of the polynomial $p_n(x) = (x-1)^n - 2^{-31}$, for $n = 2 : 55$, the condition number of which varies roughly from $10^4$ to $10^{32}$ at the real zero

If n is even, there are two real roots: $1 \pm 2^{-31/n}$; if n is odd, there is only one real root $1 + 2^{-31/n}$

We set the initial value $v_0 = 2$, then considering the local convergence property of Newton method, we deem that the iteration sequence will converge to the real root $\alpha = 1 + 2^{-31/n}$

Stopping criterion $|\widehat{v}_{k+1} - \widehat{v}_k| < tol = 10^{-15}$ and maximum admissible number of steps for the iterative process as $Num = 100$.

# Numerical experiments




Assumption values of three algorithms with respect to condition number. Here, Assumption 1 and 2 represent the largest $|\widehat{f'}(v) - f'(v)/f'(v)|$ and the largest $f(v)/f'(v)(v-\alpha)$ for all of the iterates $v$ with respect to some condition number, respectively; Assumption 3 represents the summation of Assumption 2 and double Assumption 1; Assumption 4 represents the smallest $f(v_*)/f'(v_*)(v_* - \alpha)$ with respect to some condition number

# Numerical experiments

Simple real zero of the expanded form of the polynomial $p_n(x) = (x-1)^n - 2^{-31}$, for $n = 2 : 55$