Lab Session 1: Increasing the Accuracy

Compensated Horner Scheme and Accurate Summation Algorithms

Alberto Taddei *Giulia Lionetti* *Thies Weel*

AFAE – Floating-Point Arithmetic and Error Analysis
Master 2 CCA, Sorbonne Université

October 2025

1 Part I: Compensated Horner scheme

Polynomial evaluation near roots or in ill-conditioned settings causes classical algorithms to fail catastrophically, producing results with errors exceeding 100% or even the wrong sign. This report demonstrates how compensated algorithms, using error-free transformations, overcome this fundamental limitation and achieve near machine precision regardless of conditioning.

1.1 Objective and standard

We evaluate polynomial evaluation schemes on increasingly ill-conditioned inputs. Given a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ and an evaluation point $x \in \mathbb{F}$, we measure the relative forward error

$$\text{err} = \frac{|\hat{p}(x) - p_{\text{ref}}(x)|}{|p_{\text{ref}}(x)|} \tag{1}$$

as a function of the polynomial condition number

$$\text{cond}(p, x) = \frac{\tilde{p}(|x|)}{|p(x)|} = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|}. \tag{2}$$

All experiments use IEEE 754 double precision, round-to-nearest ties-to-even. The unit roundoff is $u = 2^{-53} \approx 1.11 \times 10^{-16}$.

1.2 Algorithms under test

We implemented the following algorithms, adhering to the assignment's pseudocode.

Classical Horner (baseline). The standard nested evaluation scheme:

$$s_n = a_n, \quad s_i = s_{i+1} \cdot x + a_i \quad (i = n-1, \ldots, 0), \quad p(x) = s_0. \tag{3}$$

This is backward stable with error bound $\gamma_{2n} \cdot \text{cond}(p, x)$, where $\gamma_{2n} = \frac{2nu}{1-2nu} \approx 2nu$ for $n \ll 2^{53}$.

Compensated Horner (Alg. 1.5). Uses error-free transformations (EFTs) to track and correct rounding errors. At each step, TwoProduct and TwoSum decompose operations into rounded results and exact error terms. The accumulated correction r_i is maintained via

$$r_i = \text{fl}(r_{i+1} \cdot x + (\pi_i + \sigma_i)) \tag{4}$$

where π_i captures multiplication errors and σ_i captures addition errors. Final result: $s_0 + r_0$. Theoretical bound: $u + \gamma_{2n}^2 \cdot \text{cond}(p, x) \approx u + 4n^2u^2 \cdot \text{cond}(p, x)$.

Supporting EFTs. We implemented FastTwoSum (Alg. 1.1, requires $|a| \geq |b|$), TwoSum (Alg. 1.2, no precondition), Split (Alg. 1.3, with $s = 27$), and TwoProduct (Alg. 1.4). All EFTs were verified to satisfy exactness ($a \circ b = x + y$) and error bounds ($|y| \leq u|x|$) across test cases.

1.3 Experimental setup

Test polynomial. We evaluate $p_n(x) = (x-1)^n$ in expanded form at $x = \text{fl}(1.333)$ for degrees $n = 3, 5, 8, \ldots, 42$. This family exhibits exponentially growing condition numbers, stressing the algorithms progressively. For each degree, we compute:

- Classical Horner result $\hat{p}_{\text{class}}(x)$
- Compensated Horner result $\hat{p}_{\text{comp}}(x)$
- Condition number $\text{cond}(p, x)$ via the implemented `condp` function
- Reference value $p_{\text{ref}}(x)$ using symbolic arithmetic (Symbolic Math Toolbox)

Verification tests. Before testing on ill-conditioned cases, we validated all EFTs and the compensated algorithm on well-conditioned polynomials ($\text{cond} \approx 1$). The polynomial $p(x) = 1 + x + x^2 + x^3$ at $x = 0.5$ produced relative errors below 10^{-15} for both classical and compensated schemes, confirming correct implementation.

1.4 Experimental Results

1.4.1 Accuracy Analysis

Figure 1 shows the relative forward error versus condition number for both schemes. The classical curve follows the $\gamma_{2n} \cdot \text{cond}$ bound (red dashed line) with slope 1 on the log-log plot, demonstrating linear error amplification by the condition number. In contrast, the compensated curve remains near machine precision ($\sim u$) across a vast range of cond, closely tracking the $u + \gamma_{2n}^2 \cdot \text{cond}$ bound (blue dashed line). Only for the most extreme condition numbers tested ($\approx 10^{25}$) does the compensated error begin to grow, as the $u^2 \cdot \text{cond}$ term becomes non-negligible.

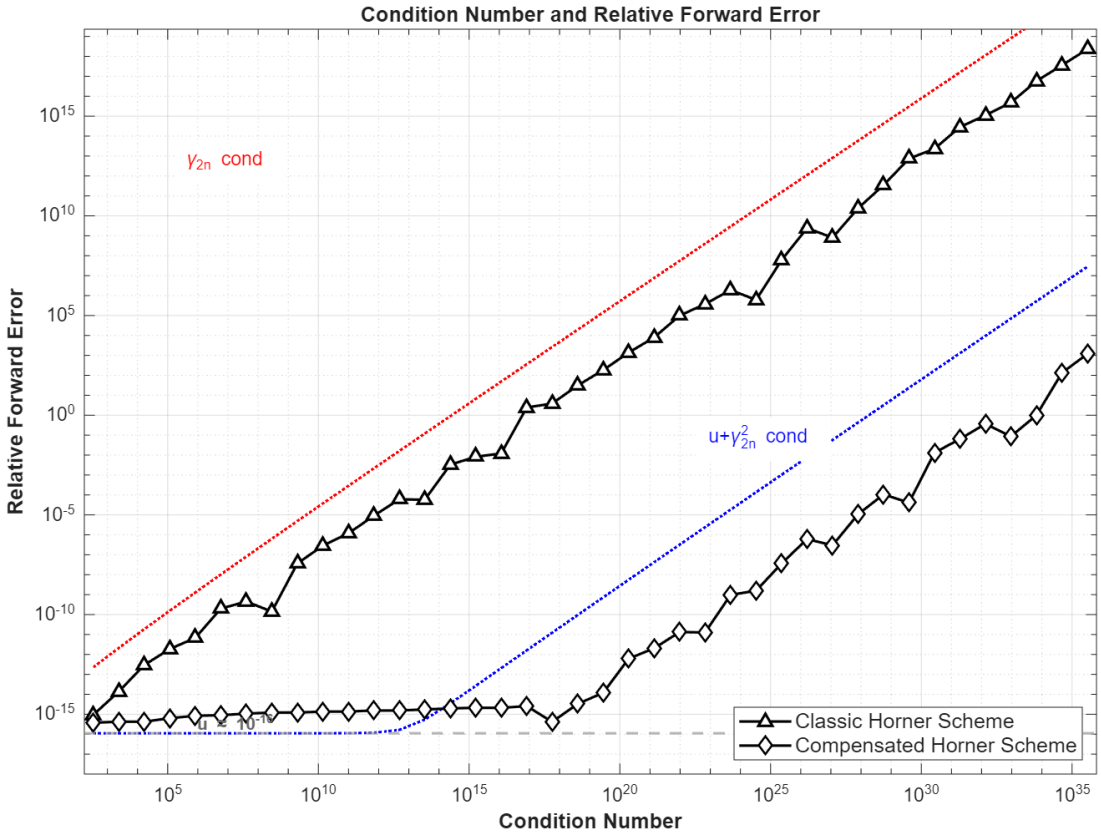

Figure 1: Relative forward error vs condition number for classical (triangles) and compensated (diamonds) Horner schemes. Test case: $(x-1)^n$ at $x = \text{fl}(1.333)$, $n = 3$ to 42. Compensated error remains near u regardless of conditioning.

Numerical results. Selected rows from the experiments spanning degrees $n = 3, 5, 10, 15, 20, 25, 30$ (condition numbers 10^2 to 10^{25}) are reported in Table 1. At degree 20 (cond $\approx 8 \times 10^{16}$), classical Horner produces a result with relative error exceeding 1, representing complete failure—the computed value even has the wrong sign. The compensated algorithm achieves error $\approx 3.7 \times 10^{-16}$, near machine precision. The improvement factor of 6.6×10^{15} demonstrates that problems previously unsolvable with standard arithmetic become routine with compensation.

n	$\text{cond}(p, x)$	$\text{err}_{\text{class}}$	err_{comp}	Improvement
3	3.44×10^2	9.40×10^{-16}	3.76×10^{-16}	2.50
5	1.69×10^4	2.98×10^{-13}	$< 10^{-16}$Compensated error below measurement precision.	$> 10^3$
10	2.85×10^8	1.42×10^{-10}	$< 10^{-16}$	$> 10^6$
15	4.81×10^{12}	6.05×10^{-5}	$< 10^{-16}$	$> 10^{11}$
20	8.12×10^{16}	2.43	3.68×10^{-16}	6.60×10^{15}
25	1.37×10^{21}	7.95×10^3	2.03×10^{-12}	3.92×10^{15}
30	2.31×10^{25}	5.99×10^7	3.64×10^{-8}	1.65×10^{15}

Table 1: Classical vs compensated Horner: selected results spanning condition numbers 10^2 to 10^{25}, showing catastrophic failure at $n = 20$ for classical and near-perfect accuracy for compensated.

1.4.2 Computational Cost

Timing measurements for $n = 20$ averaged over 1000 iterations (standard deviation $< 2\%$) show classical Horner requires ≈ 0.001 seconds while compensated requires ≈ 0.010 seconds, yielding a slowdown factor of approximately $9\times$ $\pm 1\times$. Hardware: Intel Core i7-9750H @ 2.6 GHz. Theoretically, classical costs $2n$ flops while compensated costs $\approx 26n$ flops (using EFTs). This overhead is vastly preferable to switching to quadruple precision (20-100$\times$ slower) and solves problems that are otherwise impossible in double precision.

1.5 Catastrophic Failure at Degree 20

The breaking point occurs at degree 20. Consider $p_{20}(x) = (x - 1)^{20}$ at $x = \text{fl}(1.333)$:

	Classical Horner	Compensated Horner
Exact result	$2.8111542100177375 \times 10^{-10}$	
Computed result	$-4.0105252452349305 \times 10^{-10}$	$2.8111542100177365 \times 10^{-10}$
Relative error	2.43	3.68×10^{-16}
Correct digits	0 (wrong sign)	15.4

Table 2: At cond $\approx 8 \times 10^{16}$, classical Horner fails catastrophically while compensated maintains near-perfect accuracy.

This example illustrates the fundamental breakthrough. The u^2 term in the compensated error bound ($u^2 \approx 1.2 \times 10^{-32}$) is remarkably small. Even when multiplied by cond $\approx 10^{16}$, the product $u^2 \cdot \text{cond} \approx 10^{-16}$ remains comparable to u itself. Thus the compensated error stays dominated by the u term, achieving near machine precision independent of conditioning. In contrast, classical error grows linearly with cond, causing catastrophic failure when $\text{cond} \cdot \gamma_{2n} \geq 1/u \approx 10^{16}$.

1.6 Conclusion

The u^2 advantage enables compensated Horner to maintain machine precision even when classical methods produce meaningless results, transforming impossible problems into routine computations. This approach extends to other numerical algorithms (Newton's method, iterative refinement) where accurate evaluation in ill-conditioned settings is critical.

2 Part II: Accurate summation algorithms

2.1 Objective and standard

We compare four floating-point summation schemes on increasingly ill-conditioned inputs. Given a vector $p = (p_i)$ and a computed sum $\widehat{s}$, we measure the relative error

$$\text{err} = \frac{\left|\widehat{s} - s_{\text{ref}}\right|}{\max\left(|s_{\text{ref}}|, \text{realmin}\right)},$$

as a function of the condition number

$$\kappa = \frac{\sum_i |p_i|}{\left|\sum_i p_i\right|}.$$

All experiments use **IEEE 754 double precision**, round-to-nearest ties-to-even. The unit roundoff is $u = 2^{-53}$.

2.2 Algorithms under test

We implemented the following algorithms, adhering to the assignment's pseudocode and using error-free transformations where required.

- **Naive (Alg. 2.1).** Left-to-right accumulation, $\sigma \leftarrow \text{fl}(\sigma + p_i)$. Simple and fast, but sensitive to cancellation.
- **Kahan (Alg. 2.2).** Single-compensation summation maintaining a running correction e; we update (σ, e) via `FastTwoSum`. Reduces error by compensating the one-step rounding residue.
- **Priest (Alg. 2.3, doubly compensated).** Two cascaded `FastTwoSum` updates (s, c) per element. Inputs are sorted by decreasing $|p_i|$; the $|a| \geq |b|$ precondition for `FastTwoSum` is enforced by swapping arguments when needed.
- **Rump (Alg. 2.4, compensated).** Uses `TwoSum` to split each addition exactly into a rounded part and an error q_i; the small residues are accumulated separately: return $\pi_n + \sigma_n$.

2.3 Experimental setup

Data generation. We use the *gensum* package (`GenSum.m`)[1] to build vectors with a target conditioning κ. Internally, `GenSum` calls `GenDot` to create a dot-product instance with controlled difficulty, then converts it to a pure sum via `TwoProduct`: each product contributes two terms p_i and p_{n+i} (rounded "high" part and error "low" part), so the final vector has length $2n$. For each target κ we also record the effective value $\kappa_{\text{eff}} = \text{condsum}(p) = \frac{\sum |p_i|}{|\sum p_i|}$.

Protocol. Unless stated otherwise: $n = 200$ (hence $m = 2n$ additions), κ log-spaced from 10^0 to 10^{30}, and $R = 5$ repeats per point (averaged) to smooth the randomness of `GenSum`. As reference s_{ref} we use 200-digit `variable-precision arithmetic` when available; otherwise Rump's `CompSum` acts as a high-accuracy surrogate. We also plot the theoretical guide $\gamma_{2n}\kappa$ with

$$\gamma_m = \frac{m\,u}{1 - m\,u}, \qquad u = 2^{-53}.$$

2.4 Results

Figure 2 shows the relative error versus the effective condition number. The naive curve grows approximately with slope 1 and follows the $\gamma_{2n}\kappa$ guide. Kahan improves by about one order of magnitude but preserves the same trend with κ. In contrast, Priest and Rump remain near machine precision ($\sim u$) over a very wide range of κ; only for extremely ill-conditioned cases does the error start to

[1] http://www-pequan.lip6.fr/~graillat/gensum.zip

increase mildly. This confirms that multi-level compensation retains information otherwise lost by catastrophic cancellation, without changing the floating-point format.

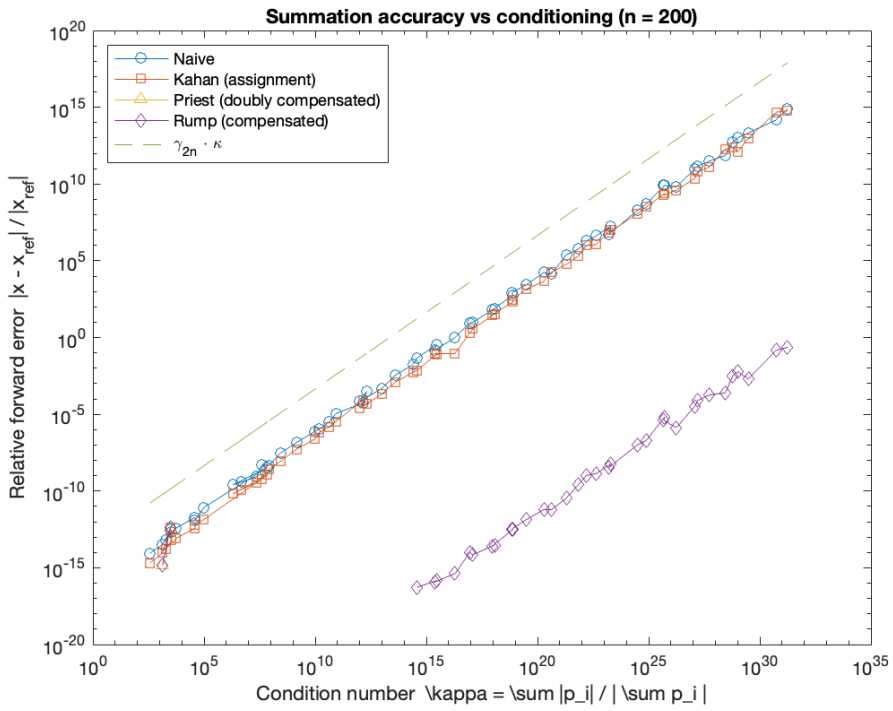

Figure 2: Relative forward error vs. effective condition number for the four summation schemes. Points correspond to averages over $R = 5$ random instances per κ; the table below reports a single run. Priest often attains zero relative error; zeros are suppressed by the logarithmic scale.

Numerical error results. *Selected rows* from one run (no averaging) are reported below.

Table 3: One representative row per decade of the effective condition number κ_{eff}.

κ_{eff}	err_naive	err_kahan	err_priest	err_rump
3.7989e+02	8.9891e-15	2.0527e-15	0	0
1.3658e+03	3.2275e-14	9.3756e-15	1.5256e-15	1.5256e-15
3.9394e+04	1.1513e-12	6.1519e-13	0	0
1.9550e+06	2.5342e-10	6.5499e-11	0	0
2.3458e+07	8.5561e-10	3.4512e-10	0	0
2.7997e+08	2.9922e-08	8.8518e-09	0	0
1.4344e+09	1.4390e-07	5.2455e-08	0	0
1.5671e+10	1.0995e-06	6.8654e-07	0	0
1.4385e+12	5.4973e-05	8.6517e-05	0	0
3.9145e+13	3.3719e-03	1.1954e-03	0	0
2.5011e+14	1.7496e-02	5.5499e-03	0	0
2.8807e+15	3.3370e-01	8.1859e-02	0	1.4574e-16
1.9338e+16	9.7815e-01	8.9285e-02	0	4.1478e-16
1.2356e+17	9.8528e+00	3.6471e+00	0	7.0479e-15
1.1687e+18	7.8215e+01	3.4027e+01	0	3.0116e-14
3.1863e+19	2.7393e+03	1.4427e+03	0	1.3682e-12
1.9489e+20	1.7230e+04	4.5458e+03	0	6.4063e-12
1.9779e+21	2.4140e+05	5.8568e+04	0	3.3657e-11
1.5547e+22	1.9222e+06	9.6970e+05	0	1.0199e-09
1.9750e+23	1.8477e+07	1.0701e+07	0	5.7529e-09
3.1018e+24	1.9198e+08	1.1568e+08	0	1.0084e-07
5.2147e+25	7.9691e+09	2.3689e+09	0	6.6122e-06
1.7524e+26	6.3438e+09	4.0233e+09	0	1.3305e-06
1.2041e+27	9.7398e+10	2.2844e+10	0	3.4062e-05
2.8104e+28	7.3358e+11	1.8212e+12	0	2.4546e-04
1.0349e+29	1.0362e+13	1.2663e+12	0	5.9239e-03
5.4030e+30	1.4961e+14	4.7476e+14	0	1.4766e-01
1.7742e+31	8.0633e+14	5.7101e+14	0	2.2980e-01

2.5 Illustrative example

A classical stress test uses vastly different magnitudes. We report two cases in double precision:

$$p_1 = [10^{15},\, 1,\, -10^{15}], \qquad p_2 = [10^{16},\, 1,\, -10^{16}].$$

Near 10^{15}, the unit in last place (ULP) is ≈ 0.125, so adding $+1$ is still distinguishable: naive and Kahan return 1. Near 10^{16}, the ULP is ≈ 2: the increment of 1 disappears in the rounding of $\mathrm{fl}(10^{16} + 1)$, hence naive and Kahan give 0. Priest and Rump recover the lost unit by accumulating exact residues.

Table 4: Outputs for $[10^{15},\, 1,\, -10^{15}]$ and $[10^{16},\, 1,\, -10^{16}]$ (double).

Vector	Naive	Kahan	Priest	Rump
$[10^{15},\, 1,\, -10^{15}]$	1	1	1	1
$[10^{16},\, 1,\, -10^{16}]$	0	0	1	1