For this practical work, one can use either MATLAB or Octave. It is divided into two parts. The first one consists in the implementation of the compensated Horner scheme and in experimentaly checking that one has the compensated rule of thumb.

The second part is devoted to a comparison of different summation algorithms in terms of the accuracy of the computed results.

1 Compensated Horner scheme

Let p be a polynomial with floating-point coefficients $p(x) = \sum_{i=0}^{n} a_i x^i$.

1) Implement the classic Horner scheme for polynomial evaluation. You can test your function by comparing with `polyval` in MATLAB or Octave.

2) Implement the Error-Free Transformations (EFT).

Algorithm 1.1. EFT for the summation of two floating-point numbers with $|a| \geq |b|$

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

Algorithm 1.2. EFT for the summation of two floating-point numbers

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

Algorithm 1.3. Error-free split of a floating point number into two parts

function $[x, y] = \mathtt{Split}(a)$
  $\mathtt{factor} = \mathrm{fl}(2^s + 1)$ % $s = 27$
  $c = \mathrm{fl}(\mathtt{factor} \cdot a)$
  $x = \mathrm{fl}(c - (c - a))$
  $y = \mathrm{fl}(a - x)$

Algorithm 1.4. EFT of the product of two floating-point numbers

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

3) Implement the compensated Horner scheme.

Algorithm 1.5. Compensated Horner scheme

function $\mathtt{res} = \mathtt{CompensatedHorner}(p, x)$
  $s_n = a_n$
  $r_n = 0$
  for $i = n - 1 : -1 : 0$
    $[p_i, \pi_i] = \mathtt{TwoProduct}(s_{i+1}, x)$

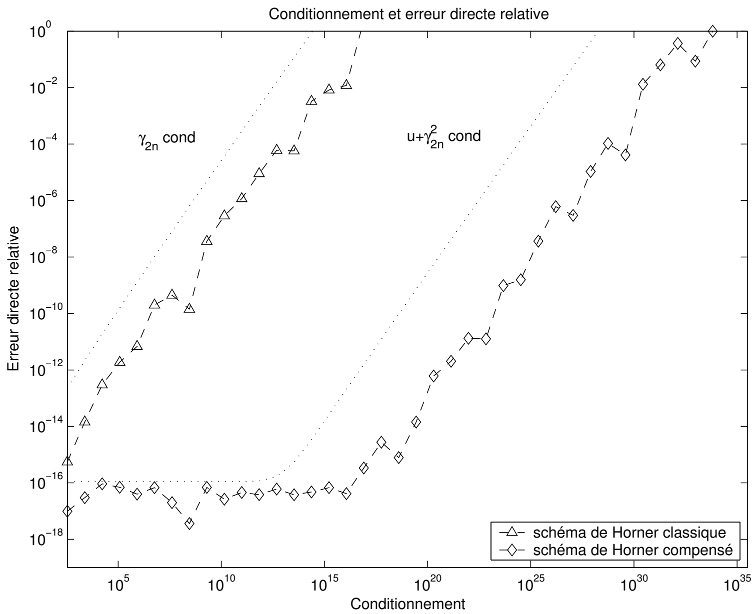
$[s_i, \sigma_i] = \mathtt{TwoSum}(p_i, a_i)$
$r_i = \mathrm{fl}((r_{i+1} \cdot x + (\pi_i + \sigma_i))$
end
$\mathtt{res} = s_0 + r_0$

4) Implement the Horner scheme in exact arithmetic. You can use the "Symbolic Math Toolbox" and in particular `sym` in MATLAB or the "symbolic" package in Octave.

5) Implement a function `condp` that computes the condition number for the evaluation of a polynomial p in x. The condition number is defined by

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

6) You will test your functions with the polynomials $p_n(x) = (x-1)^n$ with $x = \mathrm{fl}(1.333)$ and n varying from 3 to 42 in order to obtain the following picture:



2 Accurate summation algorithms

The purpose is to compare the accuracy of different algorithms for summation.

1) Implement the following summation algorithms:

Algorithm 2.1. Classic recursive summation algorithm

function $\mathtt{res} = \mathtt{Sum}(p)$
$\sigma = 0;$
for $i = 1 : n$
$\sigma = \mathrm{fl}(\sigma + p_i)$
$\mathtt{res} = \sigma$

Algorithm 2.2. Kahan's summation algorithm

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

Algorithm 2.3. Priest's doubly compensated summation algorithm

function $\texttt{res} = \texttt{DCompSum}(p)$
  we sort the p_i such that $|p_1| \geq |p_2| \geq \cdots \geq |p_n|$
  $s = 0$
  $c = 0$
  for $i = 1 : n$
    $[y, u] = \texttt{FastTwoSum}(c, p_i)$
    $[t, v] = \texttt{FastTwoSum}(s, y)$
    $z = u + v$
    $[s, c] = \texttt{FastTwoSum}(t, z)$
  $\texttt{res} = s$

Algorithm 2.4. Rump's compensated summation algorithm

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

2) Study the accuracy of those different algorithms in function of the condition number of the sum [1].

1. A MATLAB generator of ill-conditioned sum can be found here:
`http://www-pequan.lip6.fr/~graillat/gensum.zip`