Corrections dans code_aster suite à l'application de VERROU

31/03/22

Bruno Lathuilière

EDF R&D

Plan

1. Quelques corrections dans code_aster
   - Corrections par reformulation
   - Corrections par algorithmes compensés
2. REX sur les outils pour la localisation

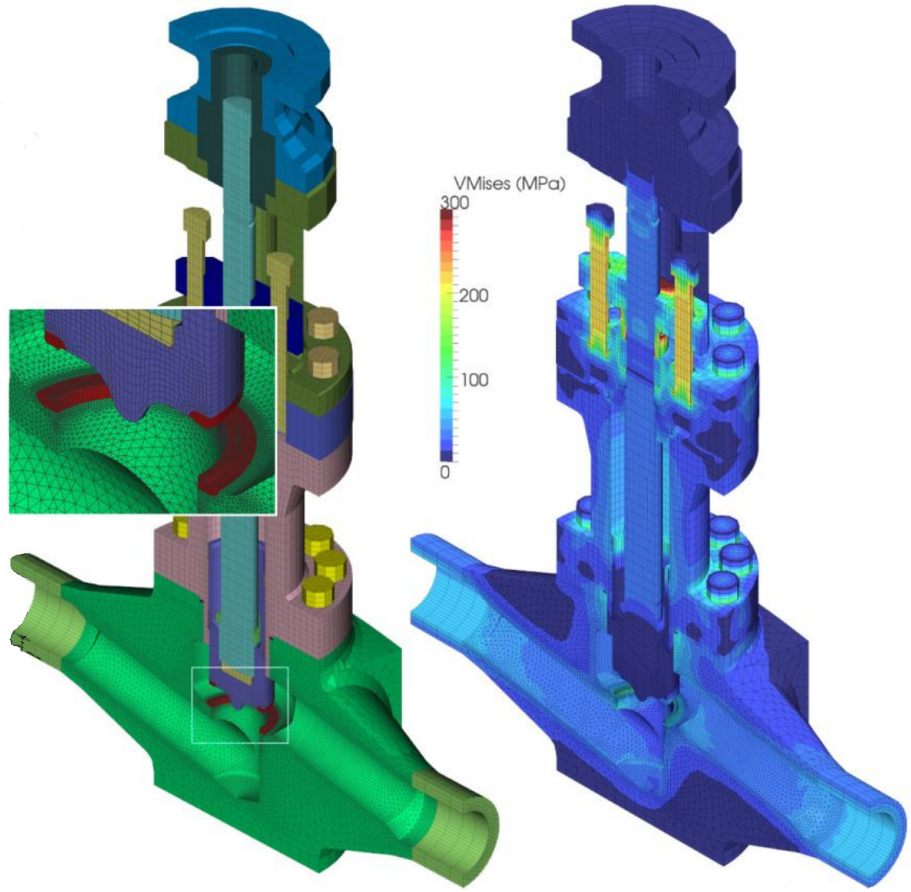
Contexte de l'étude sur code_aster

Mécanique au sens large

- Sismique
- Acoustique
- Thermique

Code_Aster

- 1.5M lignes de code
- Fortran 90, C, C++, Python
- Nombreuses dépendances :
  - solveurs linéaires (MUMPS. . .)
  - mailleurs et partitionneurs (Metis, Scotch. . .)



Intégrer Verrou à la base de non-régression

- Passer la base de non-régression.
- Sélection de deux cas test problématiques pour analyse.

fun1.f90 et 3 corrections ($n < 2$, $n = 3$, $n > 3$)

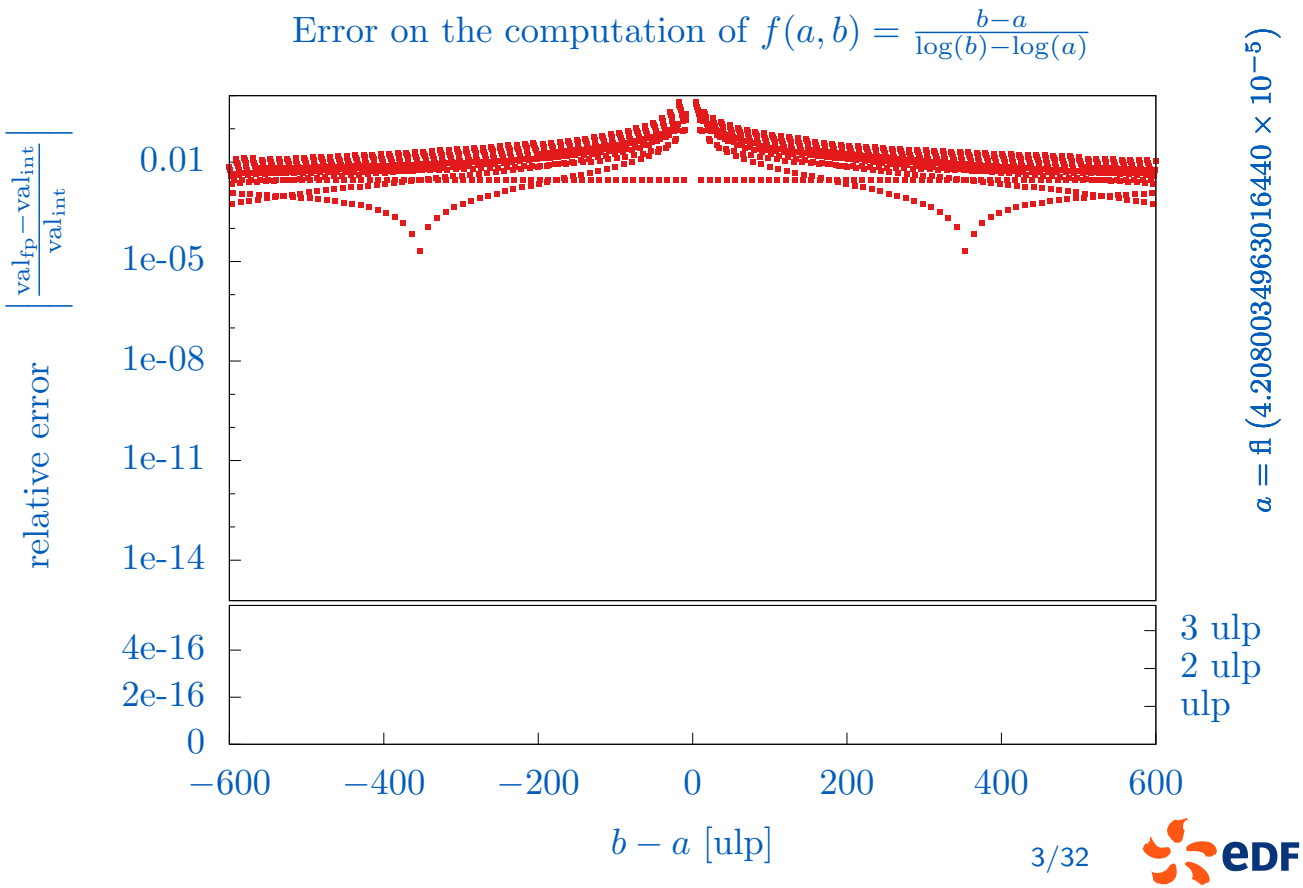
```
1       if (a1 .eq. a2) then
2           area = a1
3       else
4           if (n .lt. 2) then
5               area = (a2-a1) / (log(a2)-log(a1))
6           else if (n .eq.2) then
7 !             VARIATION HOMOTHETIQUE.
8               area = sqrt (a1*a2)
9           else if (n .eq.3) then
10              xm = 2.d0/3.d0
11              xm1 = a1 ** xm
12              xm2 = a2 ** xm
13              area = 2 * (xm1*a2 - a1*xm2 ) / (xm2-xm1)
14          else
15              xm = 1.d0 / n
16              area = (n-1)*((a2**xm)-a1**xm)
17              xm = xm-1.d0
18              area=area / ((a1**xm)-a2**xm)
19          endif
20      endif
```

Correction Formule 1

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases}$$

Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles

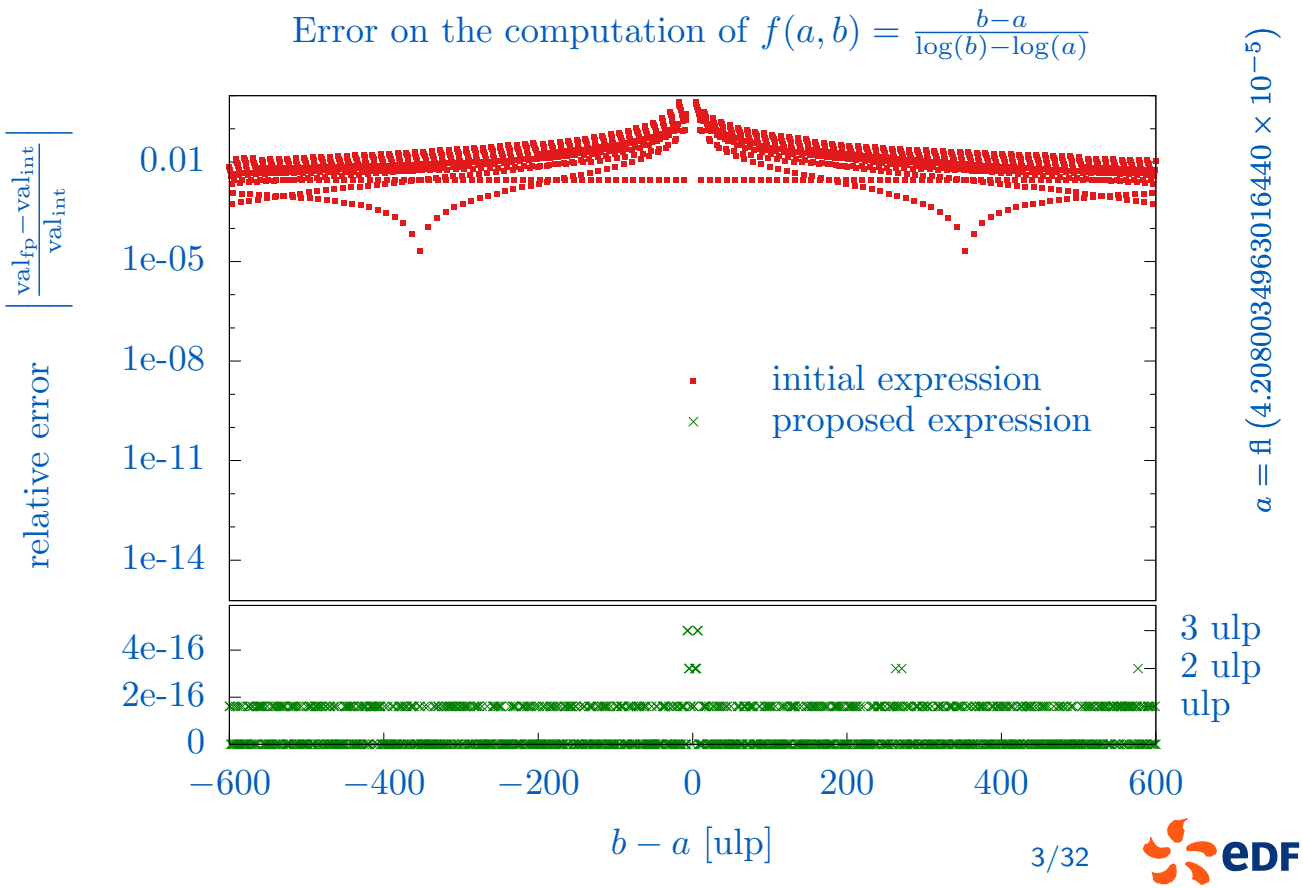


Correction Formule 1

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases} \xrightarrow[\text{manuelle}]{\text{récriture}} f(a,b) = \begin{cases} a & \text{si } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{sinon} \end{cases}$$

Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles



Correction Formule 1

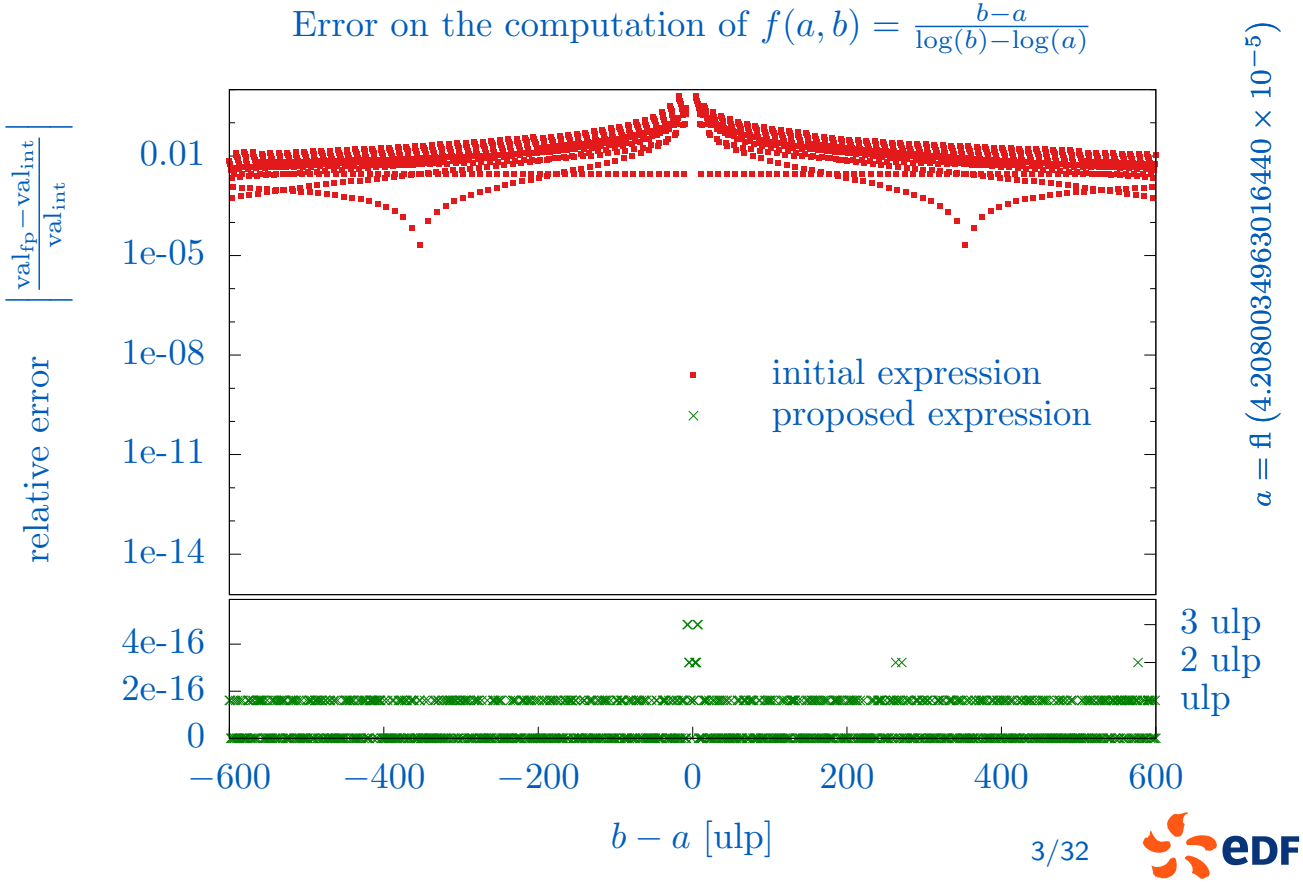
$$f(a,b) = \begin{cases} a & \text{si } a = b \\ \frac{b-a}{\log(b)-\log(a)} & \text{sinon} \end{cases} \xrightarrow[\text{manuelle}]{\text{récriture}} f(a,b) = \begin{cases} a & \text{si } a = b \\ a\,\frac{\frac{b}{a}-1}{\log(\frac{b}{a})} & \text{sinon} \end{cases}$$

Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles

Preuve

- erreur bornée par 10 ulps

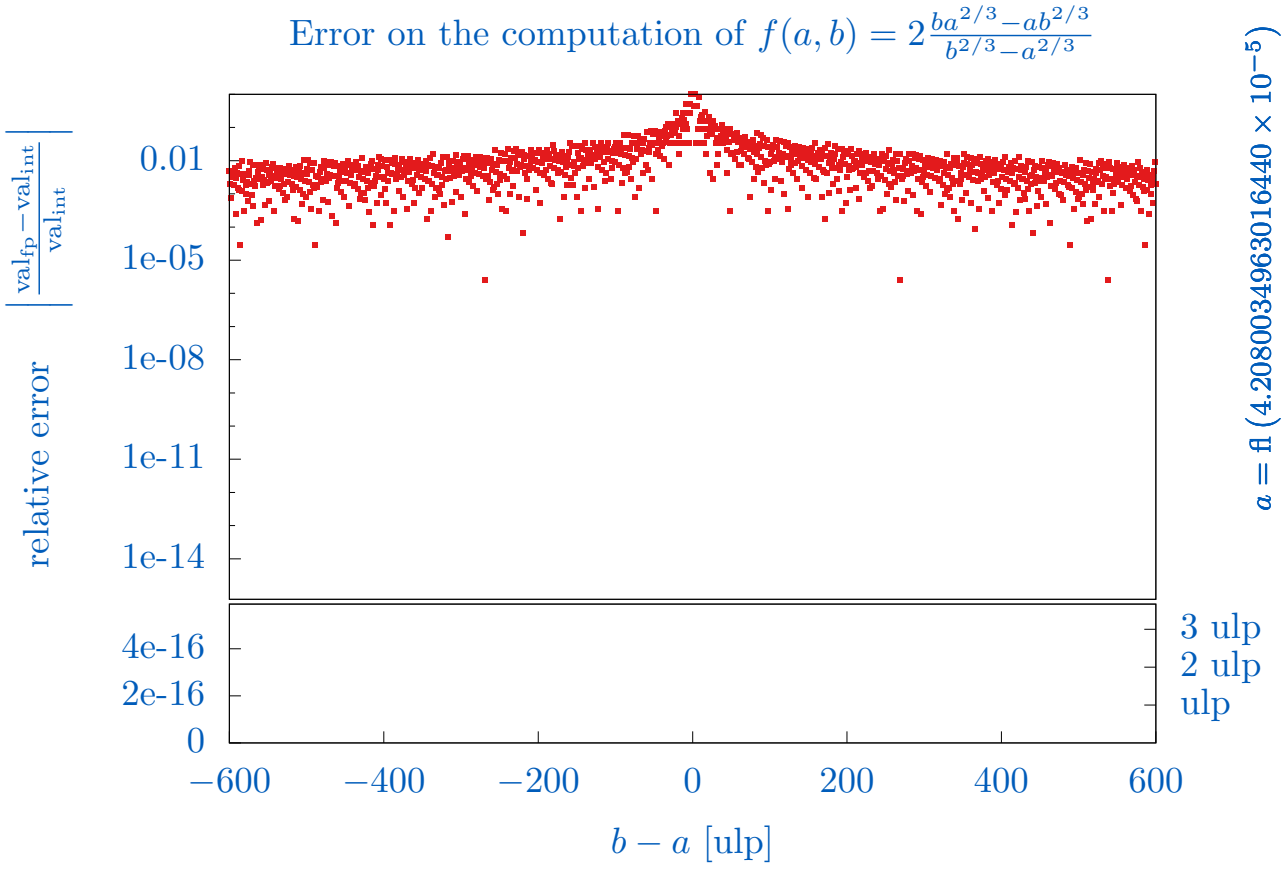


Correction Formule 2

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{sinon} \end{cases}$$

Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles

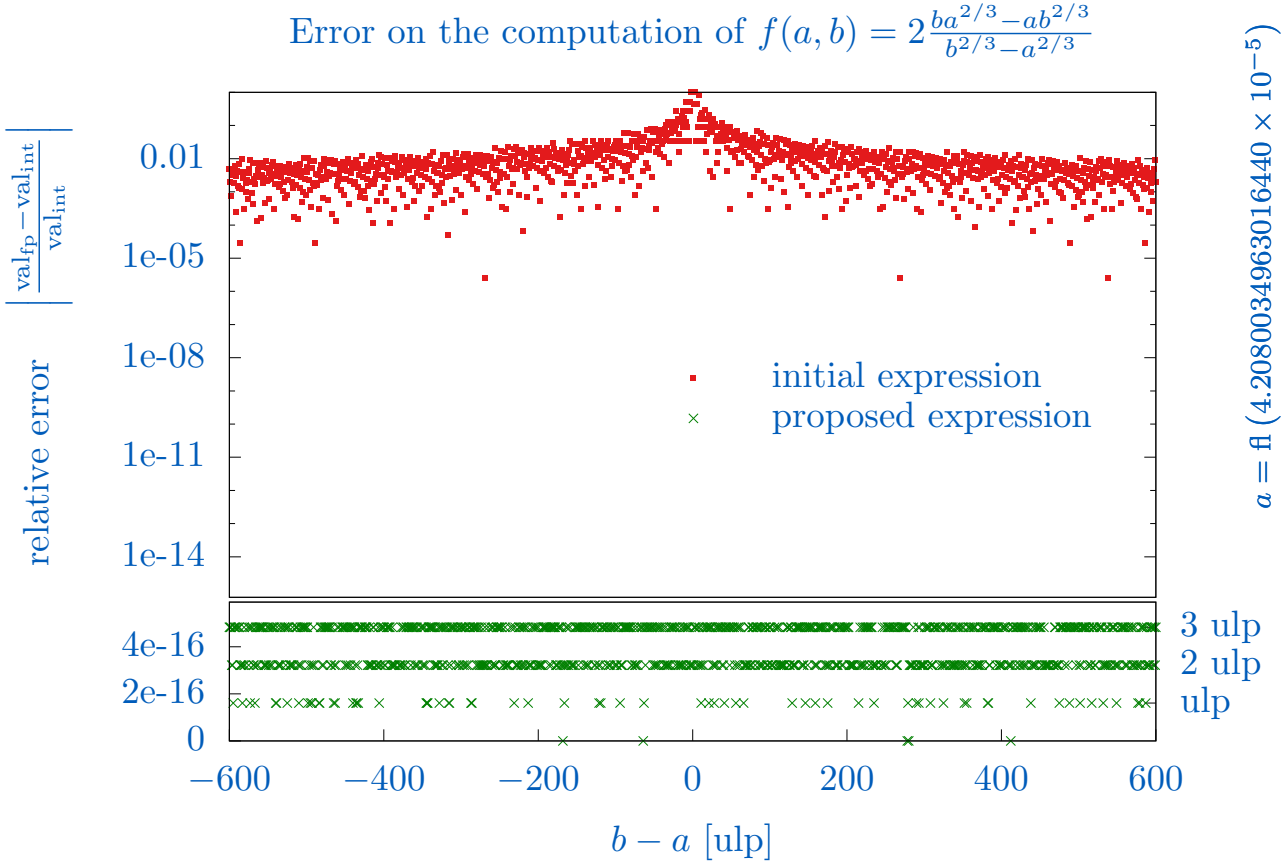


Correction Formule 2

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{sinon} \end{cases} \xrightarrow[\text{alpha}]{\text{wolfram}} f(a,b) = 2\frac{a^{2/3}b^{2/3}}{a^{1/3} + b^{1/3}}$$

Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles

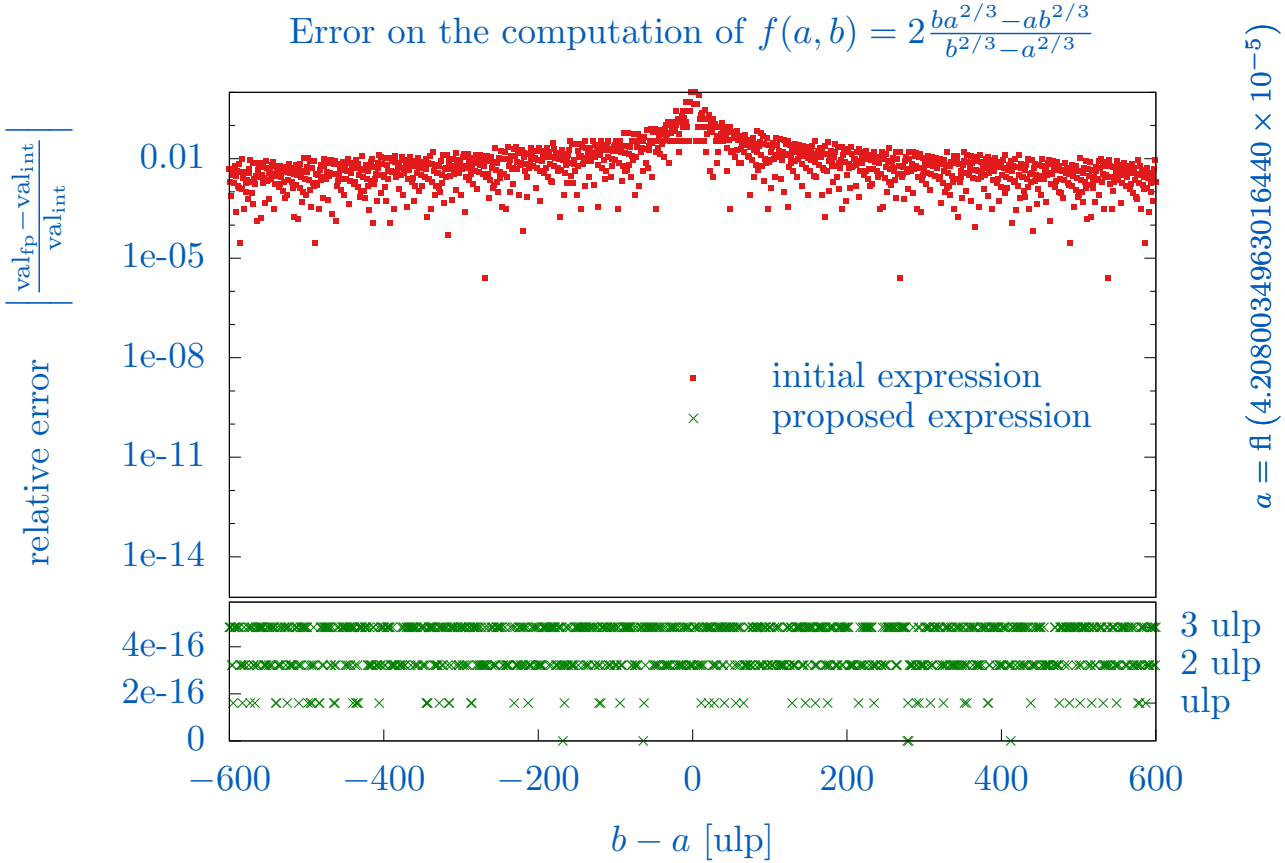


Correction Formule 2

$$f(a,b) = \begin{cases} a & \text{si } a = b \\ 2\frac{ba^{2/3} - ab^{2/3}}{b^{2/3} - a^{2/3}} & \text{sinon} \end{cases} \xrightarrow[\text{alpha}]{\text{wolfram}} f(a,b) = 2\frac{a^{2/3}b^{2/3}}{a^{1/3} + b^{1/3}}$$

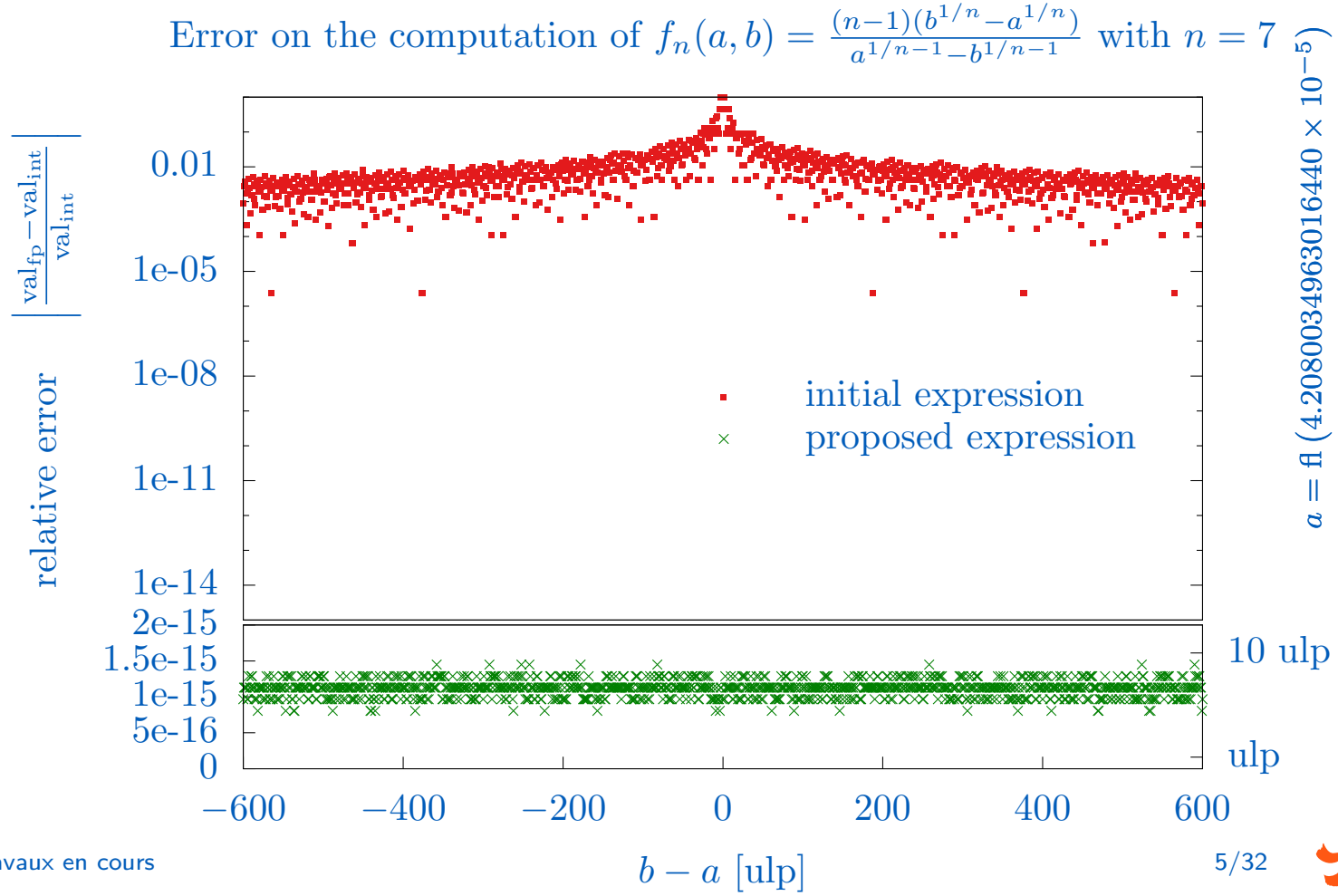
Étude empirique

- en dehors du code
- autour du point problématique
- référence = arithmétique d'intervalles



Correction Formule 3

$$f_n(a,b) = \begin{cases} a & \text{si } a = b \\ (n-1)\dfrac{b^{\frac{1}{n}} - a^{\frac{1}{n}}}{a^{\frac{1}{n}-1} - b^{\frac{1}{n}-1}} & \text{sinon} \end{cases} \xrightarrow[\text{manuelle}]{\text{récriture}} f_n(a,b) = \frac{n-1}{\sum_{i=1}^{n-1} a^{\frac{i-n}{n}} b^{\frac{-i}{n}}}$$



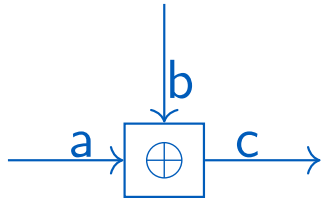
Plan

1. Quelques corrections dans code_aster
   - Corrections par reformulation
   - Corrections par algorithmes compensés
2. REX sur les outils pour la localisation

Les algorithmes compensés en 2 slides : les EFT

Soient $a, b \in \mathbb{F}^2$

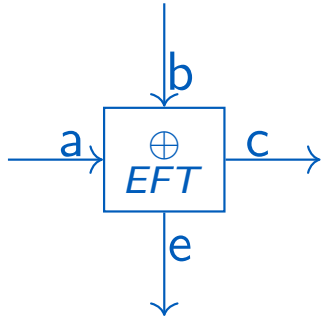
$c = a \oplus b$




```
1 c=a+b;
```

c,e=[Fast]TwoSum(a,b)

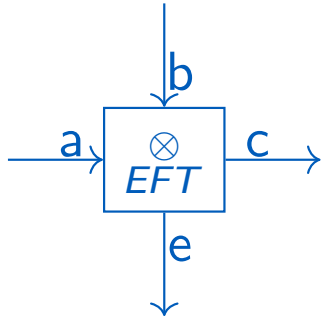
avec $\left| \begin{array}{l} c + e =_{|\mathbb{R}} a + b \\ c = a \oplus b \end{array} \right.$




```
1 if  abs(a)< abs(b){
2     swap(a,b);
3 }
4 c=a+b;
5 e=(b-(c-a));
```

c,e=TwoProd(a,b)

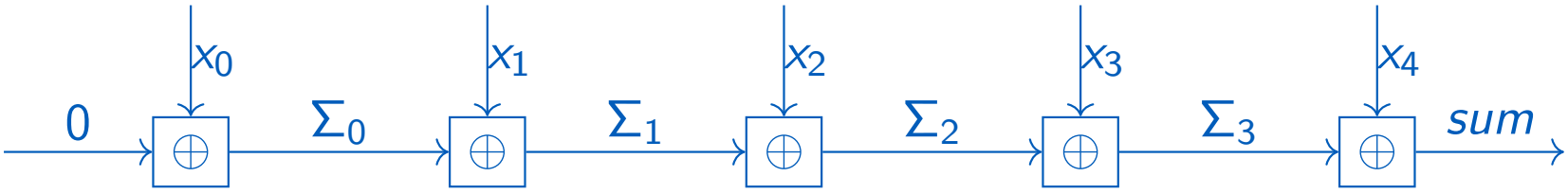
avec $\left| \begin{array}{l} c + e =_{|\mathbb{R}} a \times b \\ c = a \otimes b \end{array} \right.$




```
1 c=a*b;
2 e=fma(a, b, -c);
```

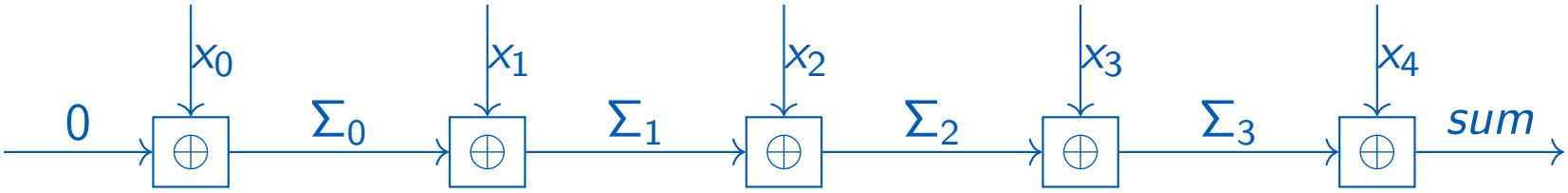
Les algorithmes compensés en 2 slides : la somme

L'algorithme de somme naïve :

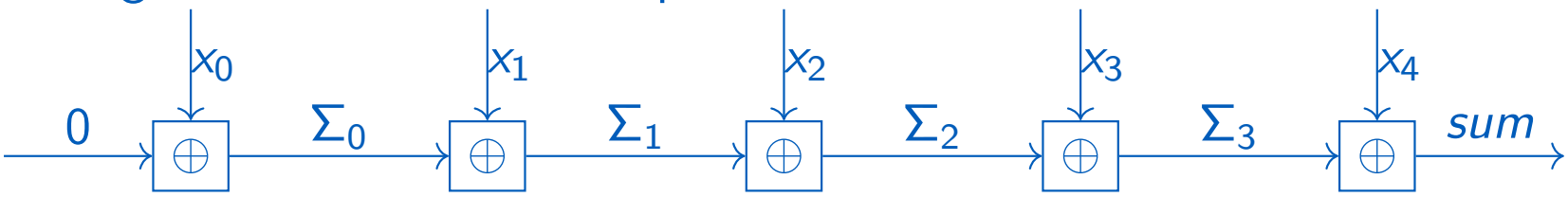


Les algorithmes compensés en 2 slides : la somme

L'algorithme de somme naïve :

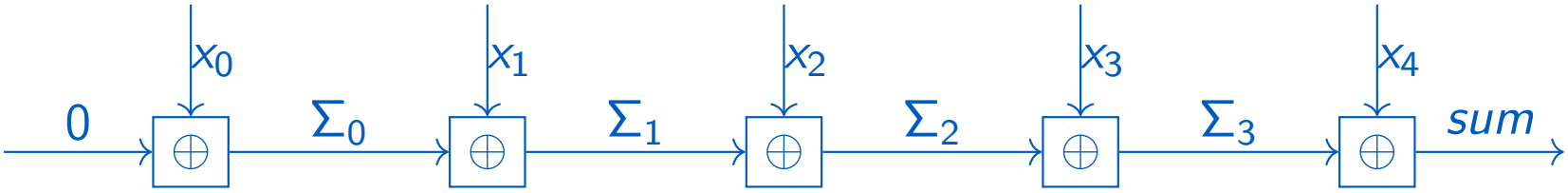


Un algorithme de somme compensée:

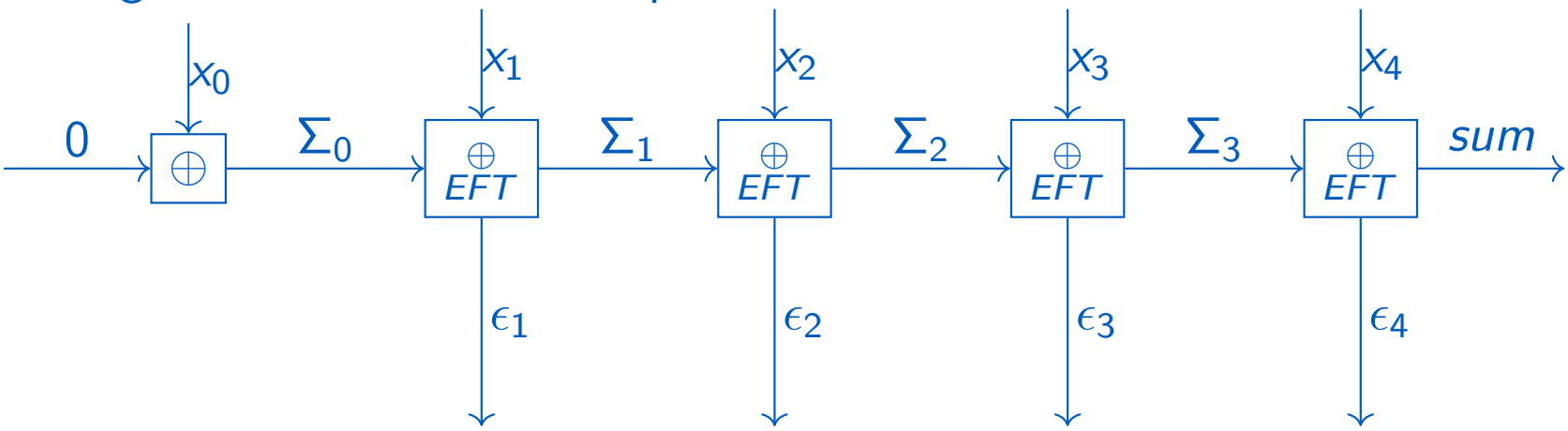


Les algorithmes compensés en 2 slides : la somme

L'algorithme de somme naïve :

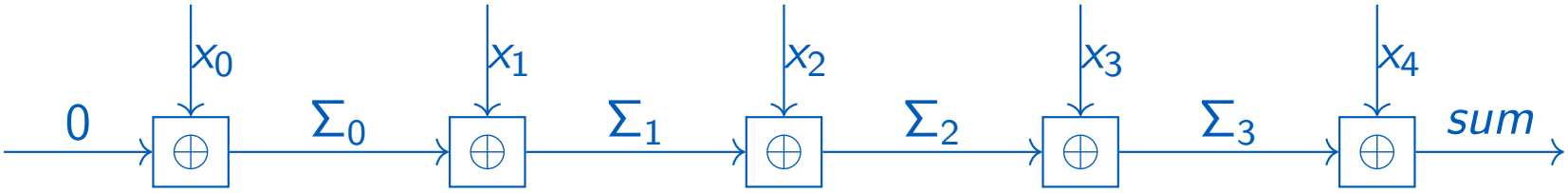


Un algorithme de somme compensée:

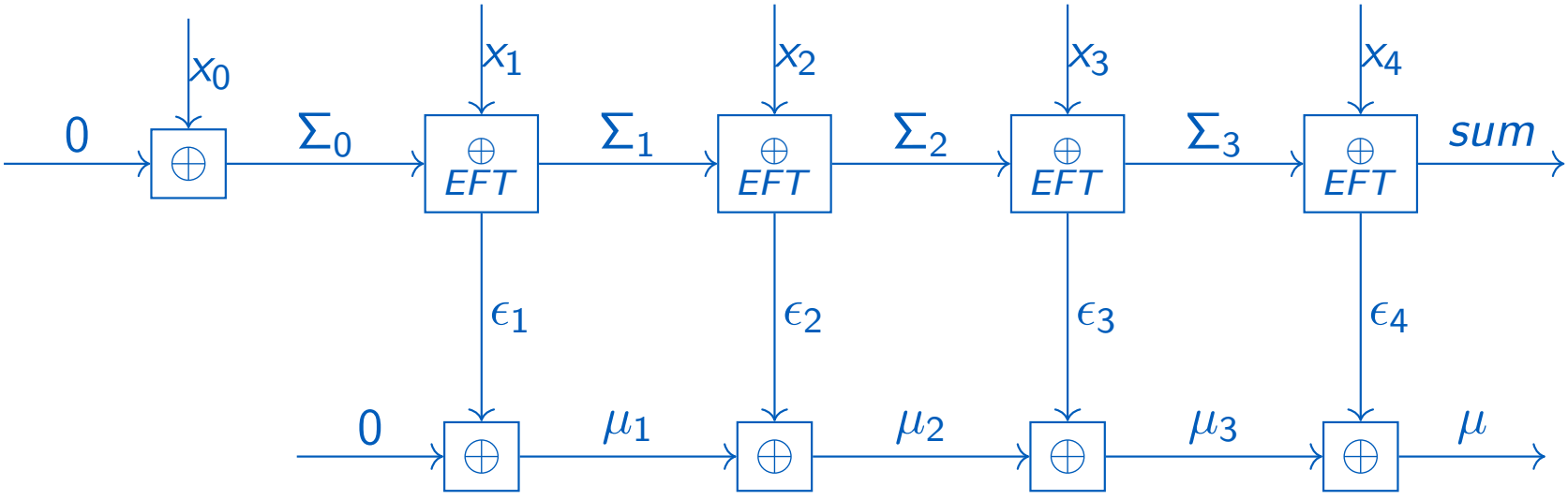


Les algorithmes compensés en 2 slides : la somme

L'algorithme de somme naïve :

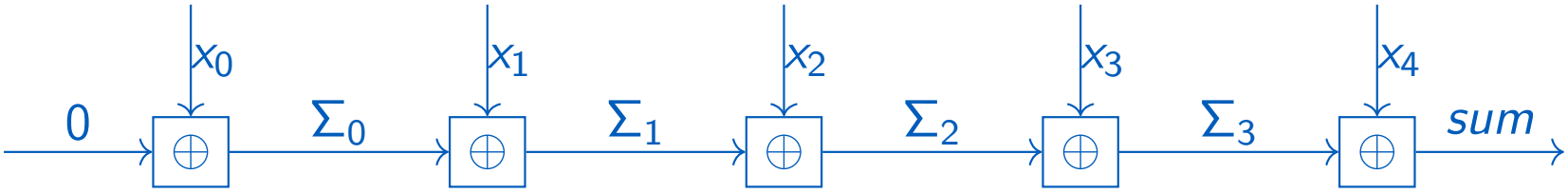


Un algorithme de somme compensée:

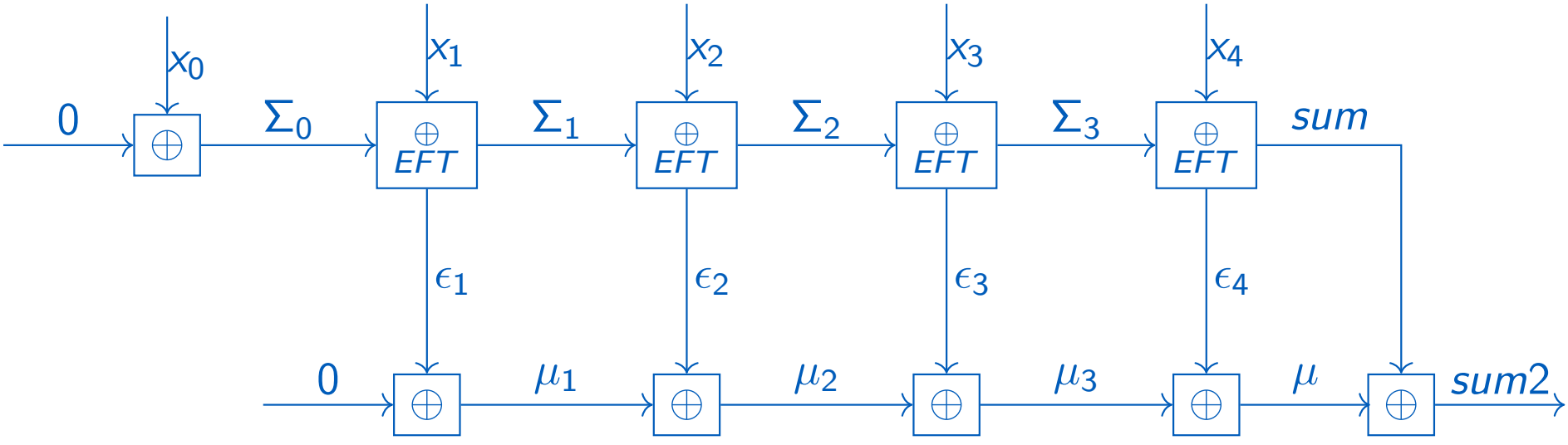


Les algorithmes compensés en 2 slides : la somme

L'algorithme de somme naïve :



Un algorithme de somme compensée:



Corrections rc32my.F90 : contexte

1. Passage avec Verrou d'une partie de la base de cas test code_aster
2. Parmi les cas test problématiques, on choisit **rccm01c**.
3. Le delta-debug pointe plusieurs lignes dont une dans **rc32my.F90**.
4. On extrait rc32my.F90 pour l'analyser.
5. Retro ingénieurie pour comprendre ce qu'on calcule.

rc32my calcule momen0 et momen1 :

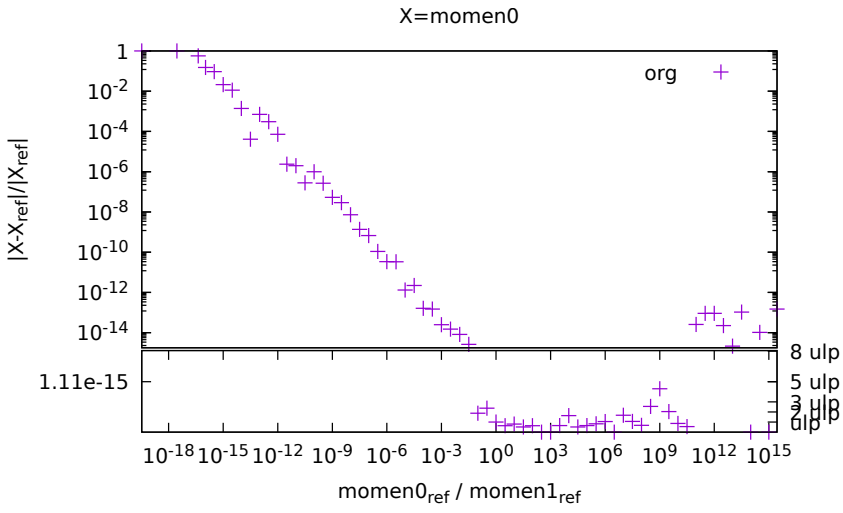
- $momen0 = \frac{1}{b-a} \int_{[a,b]} f(x)dx$
- $momen1 = \int_{[a,b]} \frac{1}{12} \left(\frac{x-a}{b-a} - \frac{1}{2} \right) f(x)dx$

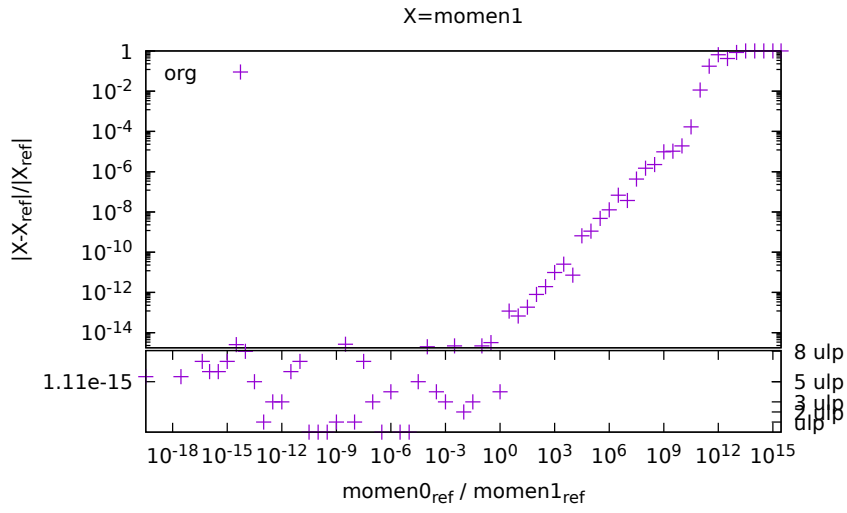
avec f définie sur un ensemble de points sur $[a, b]$ via la méthode des trapèzes pour les termes en f(x) et la méthode de simpson pour les termes en x.f(x).

rc32my.F90 => CPP [org]

```
1   const std::vector<REALTYPE>& v(input.v);
2   const std::vector<REALTYPE>& absc(input.absc);
3   REALTYPE momen0=0.,momen1=0;
4   const unsigned int size=v.size();
5   const REALTYPE l=1/(absc[size-1]-absc[0]);
6   for(size_t i=0; i< size-1; i++){
7     const REALTYPE s1 = absc[i]   - absc[0];
8     const REALTYPE s2 = absc[i+1] - absc[0];
9     const REALTYPE s12 = s2-s1;
10    const REALTYPE t1 = v[i], t2 = v[i+1];
11    const REALTYPE t12 = (t1+t2)/2.;
12    const REALTYPE smil = (s1+s2)/2.;
13    momen0 = momen0 + s12*(t1+t2);
14    momen1 = momen1 + s12/3 *(t1*s1+4*t12*smil+t2*s2);
15  }
16  momen0 = momen0*l;
17  momen1 = momen1*l*l;
18  momen0 = 0.5*momen0;
19  momen1 = 6.0*(momen1 - momen0);
20  return std::vector<REALTYPE>({momen0,momen1});
```

rc32my.F90 => CPP [org]



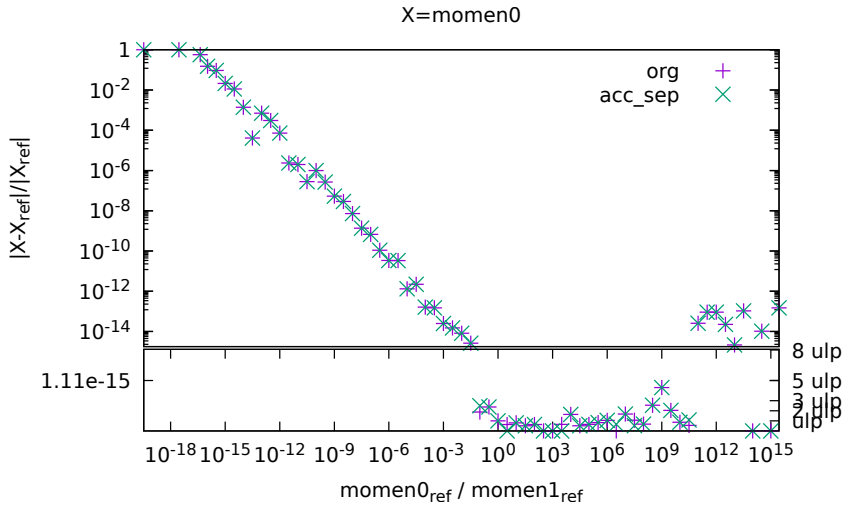


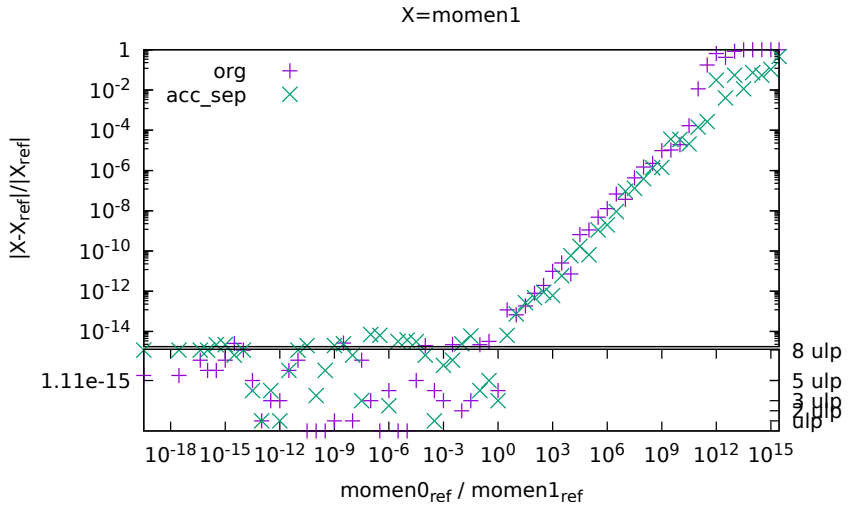
- Données d'entrée choisies avec momen1=1 et momen0 variant.
- Références obtenues via l'arithmétique d'intervalle multi-precision;

rc32my.F90 => CPP [acc_sep]

```
1   REALTYPE momen0=0.,momen1=0. ;
2   const unsigned int size=v.size();
3   const REALTYPE l=(absc[size-1]-absc[0]);
4
5   for(size_t i=0; i< size-1; i++){
6     const REALTYPE s1 = (absc[i]   - absc[0]) / l;
7     const REALTYPE s2 = (absc[i+1]- absc[0]) / l;
8     const REALTYPE s12= absc[i+1] - absc[i];
9
10    const REALTYPE t1=v[i], t2=v[i+1];
11
12    const REALTYPE momen0_i(0.5*s12*(t1+t2));
13    momen0 = momen0 + momen0_i;
14
15    const REALTYPE momen1_i = s12 ↩
          *(t1*(4.*s1+2*s2-3.)+t2*(4.*s2+2*s1-3.));
16    momen1 = momen1 + momen1_i;
17  }
18  momen0 = momen0/l;
19  momen1 = momen1/l;
```

rc32my.F90 => CPP [acc_sep]

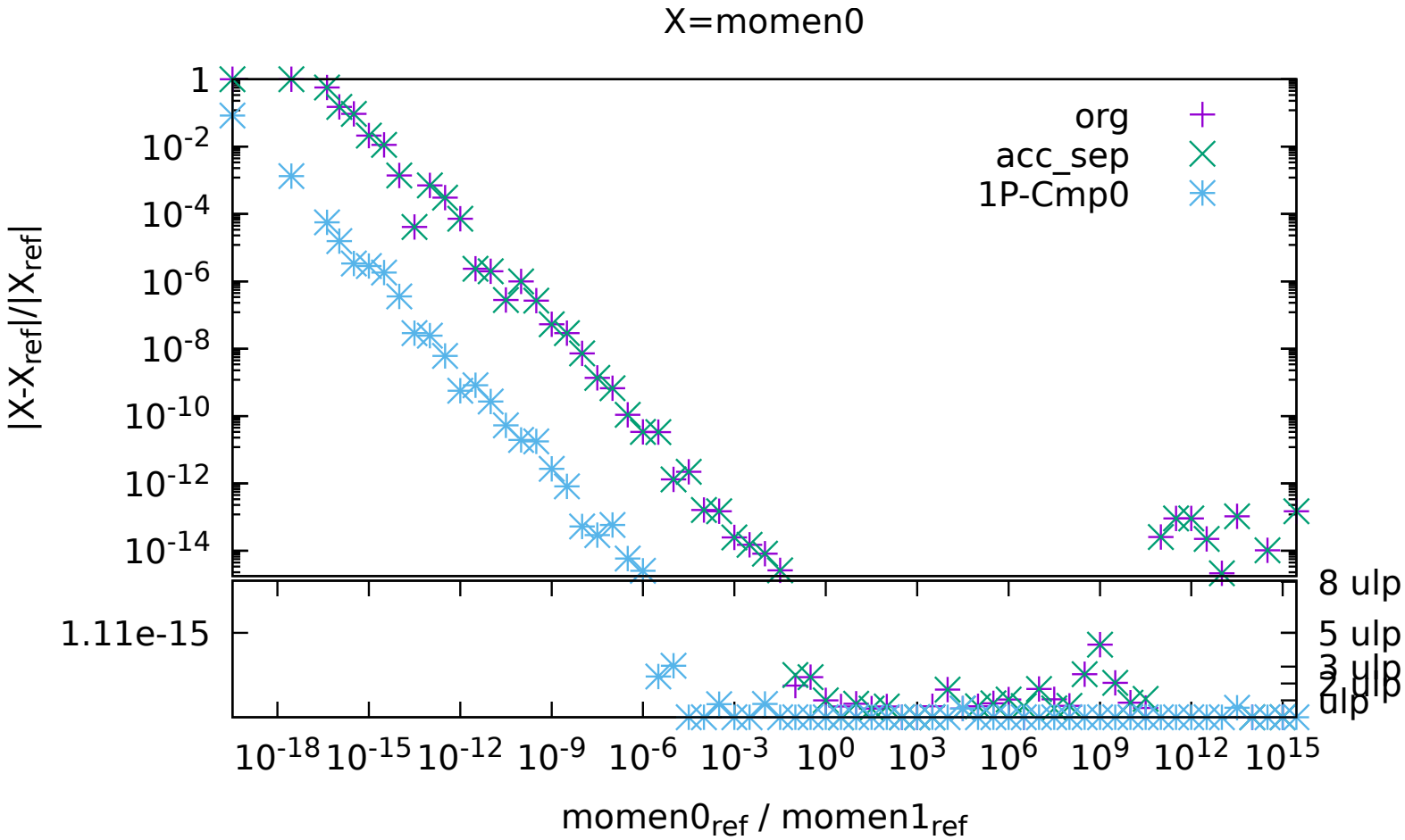




rc32my.F90 => CPP [1P-Cmp0]

```
1   REALTYPE momen0=0,momen0_lo=0.,momen1=0.;
2   for(size_t i=0; i< size-1; i++){
3     const REALTYPE s1 = (absc[i]   - absc[0])/l;
4     const REALTYPE s2 = (absc[i+1] - absc[0])/l;
5     const REALTYPE s12= absc[i+1] - absc[i];
6     const REALTYPE t1 = v[i], t2 = v[i+1];
7
8     const REALTYPE momen0_i = 0.5*s12*(t1+t2);
9     REALTYPE err;
10    EFT::twoSum(momen0, momen0_i, momen0, err);
11    momen0_lo += err;
12
13    const REALTYPE momen1_i = s12 ↩
          *(t1*(4.*s1+2*s2-3.)+t2*(4.*s2+2*s1-3.));
14    momen1 += momen1_i;
15  }
16  momen0 += momen0_lo;
17
18  momen0 = momen0/l;
19  momen1 = momen1/l;
```

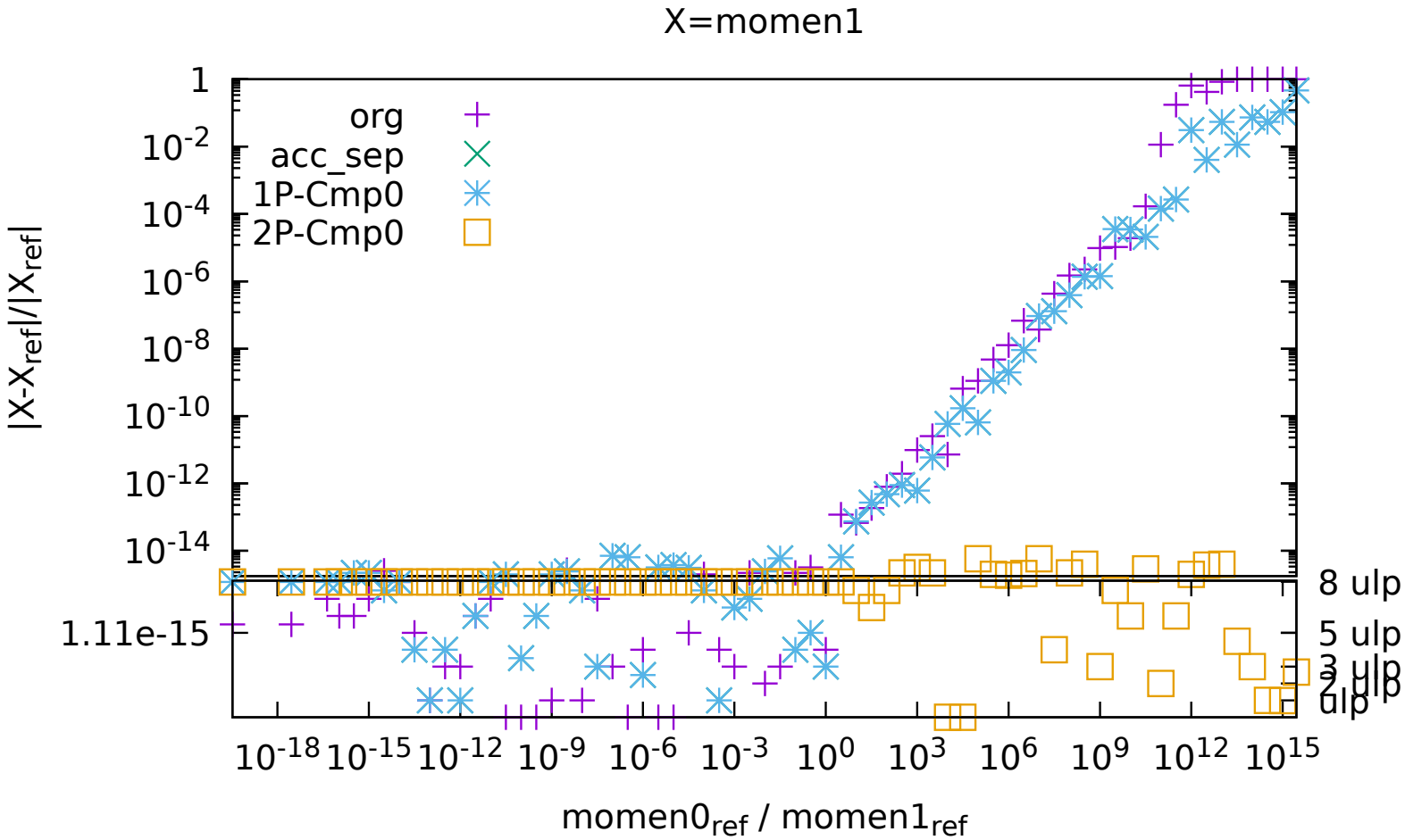
rc32my.F90 => CPP [1P-Cmp0]



rc32my.F90 => CPP [2P-Cmp0]

```
1   REALTYPE momen0=0., momen0_lo=0.,momen1=0.;
2   for(size_t i=0; i< size-1; i++){
3     const REALTYPE s12=absc[i+1] -absc[i];
4     const REALTYPE t1=v[i],t2=v[i+1] ;
5     const REALTYPE momen0_i(REALTYPE(0.5)*s12*(t1+t2));
6     EFT::twoSum(momen0, momen0_i, momen0, err);
7     momen0_lo += err;
8   }
9   momen0+=momen0_lo; momen0=momen0/l;
10  for(size_t i=0; i< size-1; i++){
11    const REALTYPE s1 = ... , s2 = ..., s12 = ...;
12    const REALTYPE t1 = v[i]   - momen0;
13    const REALTYPE t2 = v[i+1]- momen0;
14    const REALTYPE momen1_i = s12 ↩
          *(t1*(4.*s1+2*s2-3.)+t2*(4.*s2+2*s1-3.));
15    momen1 += momen1_i;
16  }
17  momen1 = momen1/l;
```

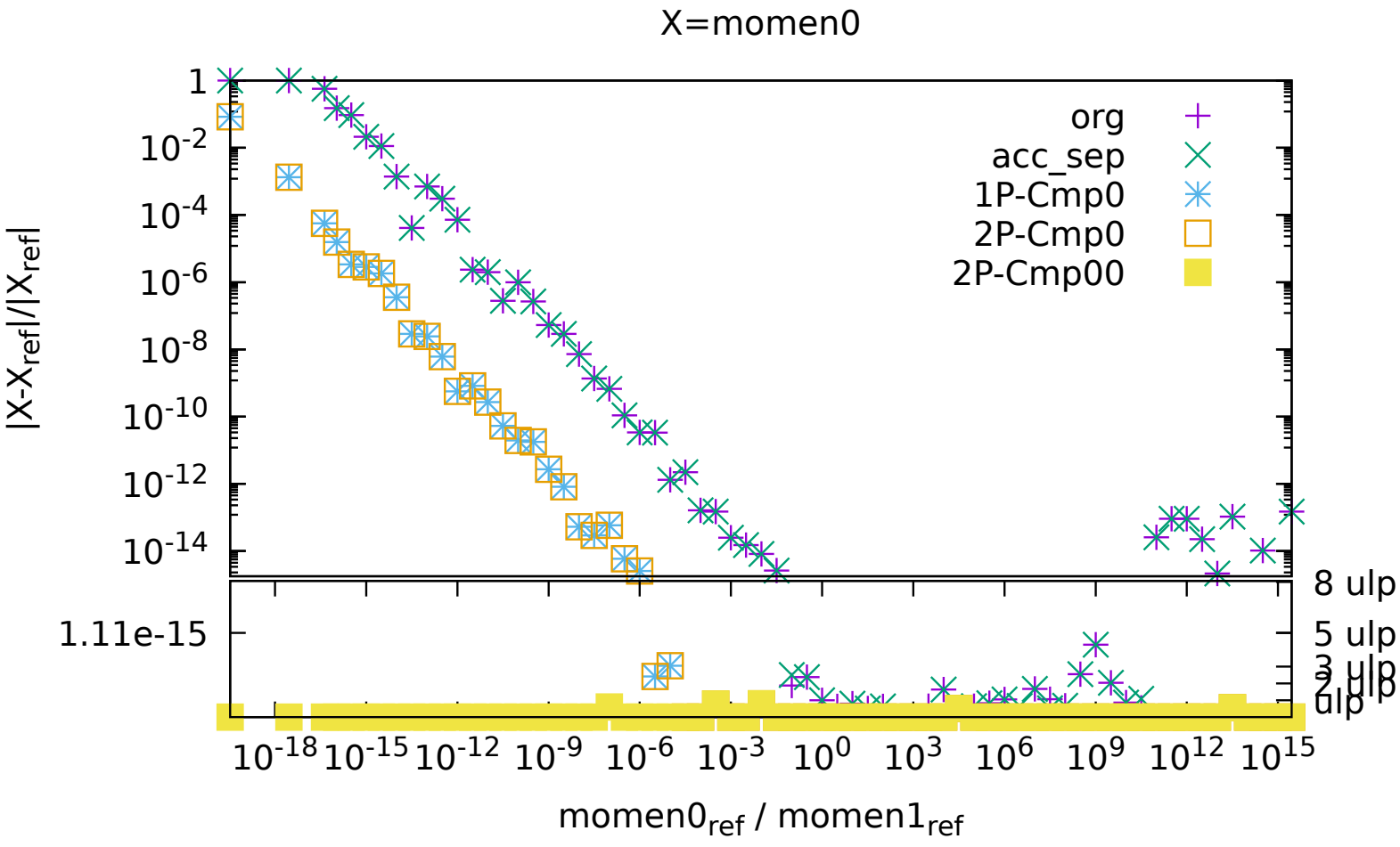
rc32my.F90 => CPP [2P-Cmp0]



rc32my.F90 => CPP [2P-Cmp00]

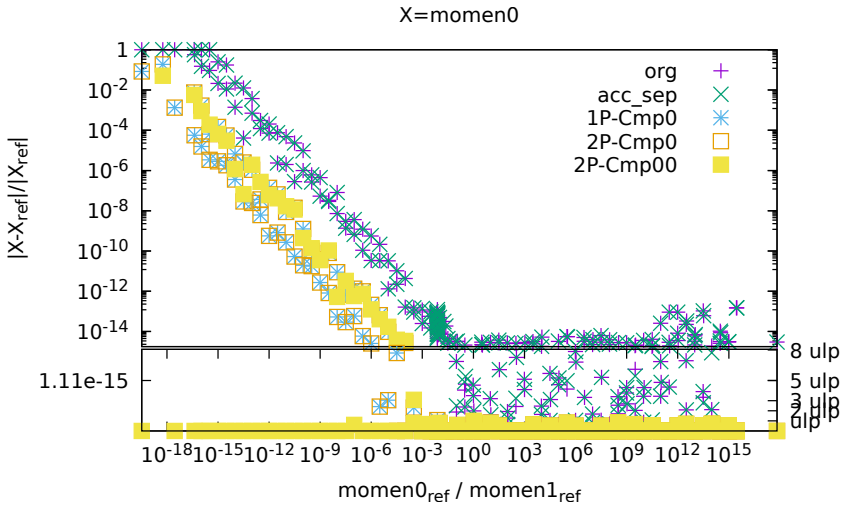
```
1  REALTYPE momen0=0., momen0_lo=0.,momen0_lo2=0.,momen1=0.;
2  for(size_t i=0; i< size-1; i++){
3      const REALTYPE s12 = absc[i+1] -absc[i];
4      const REALTYPE t1 = v[i], t2 = v[i+1];
5
6      REALTYPE t1pt2_hi, t1pt2_lo;
7      EFT::twoSum(t1, t2, t1pt2_hi, t1pt2_lo);
8      const REALTYPE s12div2(0.5*s12);
9      const REALTYPE momen0_i = s12div2*t1pt2_hi;
10     t1pt2_lo *= s12div2;
11
12     REALTYPE err;
13     EFT::fastTwoSum(momen0, momen0_i, momen0, err);
14     momen0_lo  += err;
15     momen0_lo2 += t1pt2_lo;
16   }
17   momen0 += (momen0_lo+momen0_lo2);
18   momen0 = momen0/l;
19   ...
```

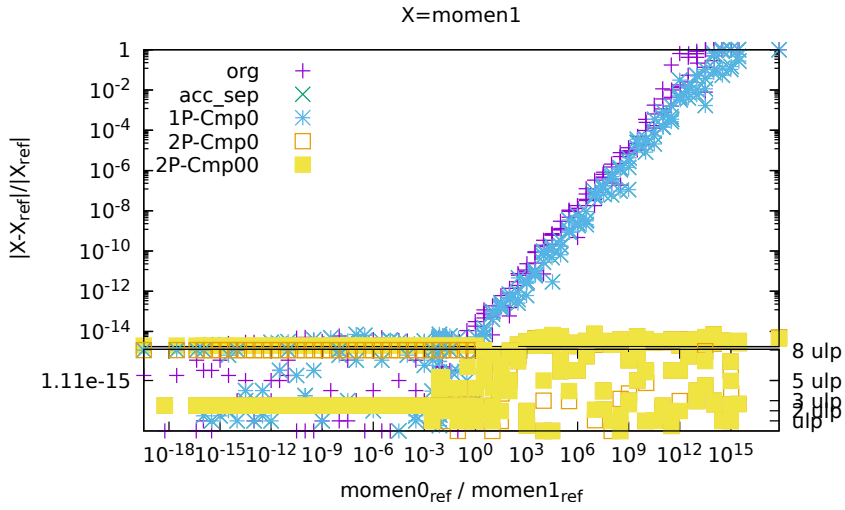
rc32my.F90 => CPP [2P-Cmp00]



Champagne?

Avec des données d'entrée bruitées

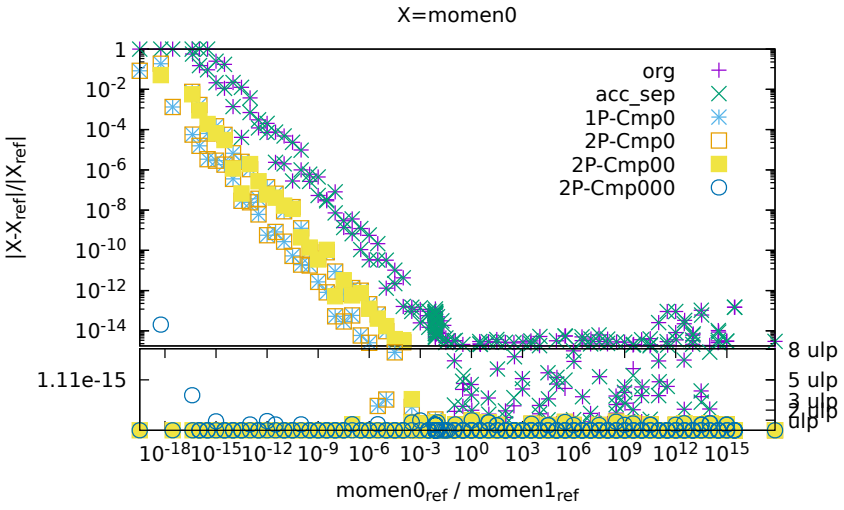


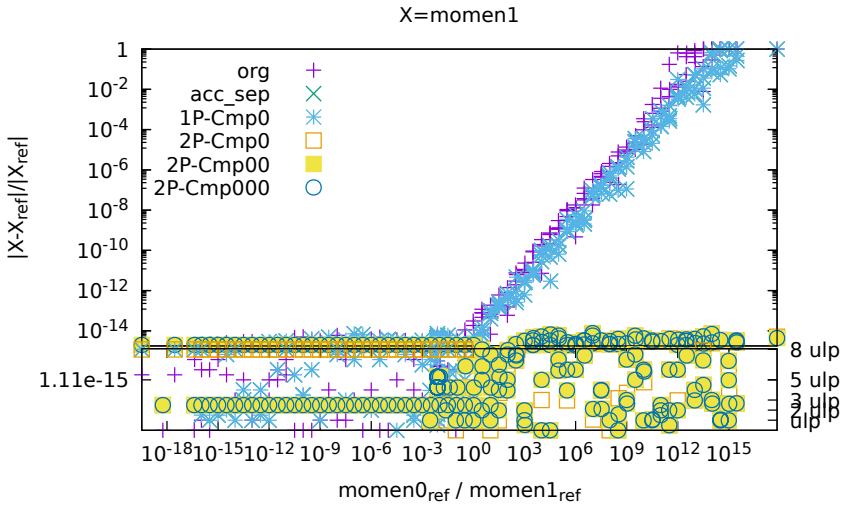


rc32my.F90 => CPP [2P-Cmp000]

```
1  for(size_t i=0; i< size-1; i++){
2    REALTYPE s12_hi, s12_lo;
3    EFT::fastTwoSum(absc[i+1],-absc[i],s12_hi,s12_lo);
4    const REALTYPE s12div2_hi = (0.5*s12_hi);
5    const REALTYPE s12div2_lo = (0.5*s12_lo);
6
7    const REALTYPE t1 = v[i], t2 = v[i+1];
8    REALTYPE t1pt2_hi, t1pt2_lo;
9    EFT::twoSum(t1, t2, t1pt2_hi, t1pt2_lo);
10
11   REALTYPE momen0_i_hi, momen0_i_lo;
12   EFT::twoProd(s12div2_hi,t1pt2_hi,momen0_i_hi,momen0_i_lo)
13   momen0_i_lo += s12div2_lo*t1pt2_hi;
14   momen0_i_lo += s12div2_hi*t1pt2_lo;
15
16   EFT::fastTwoSum(momen0, momen0_i_hi, momen0, err);
17   momen0_lo1 += err;
18   momen0_lo2 += momen_i_lo;
19 }
20 momen0 += (momen0_lo1 + momen0_lo2);
21 momen0 = momen0/l;
```

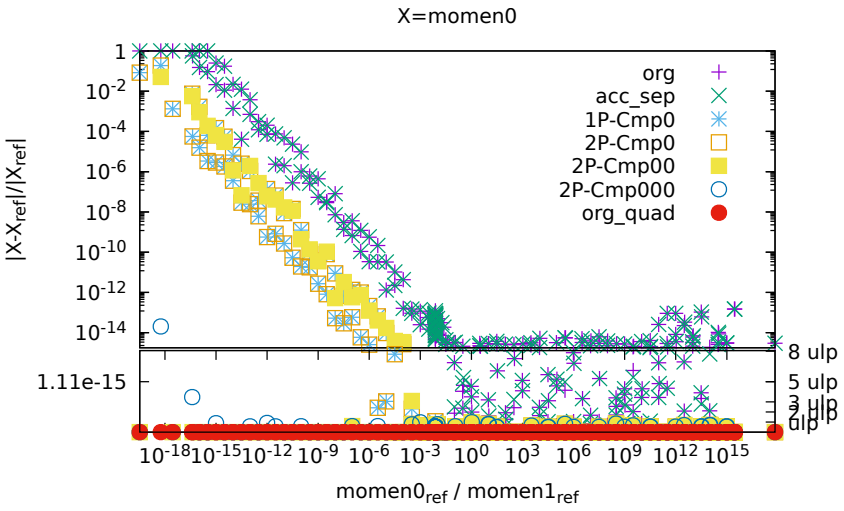
rc32my.F90 => CPP [2P-Cmp000]

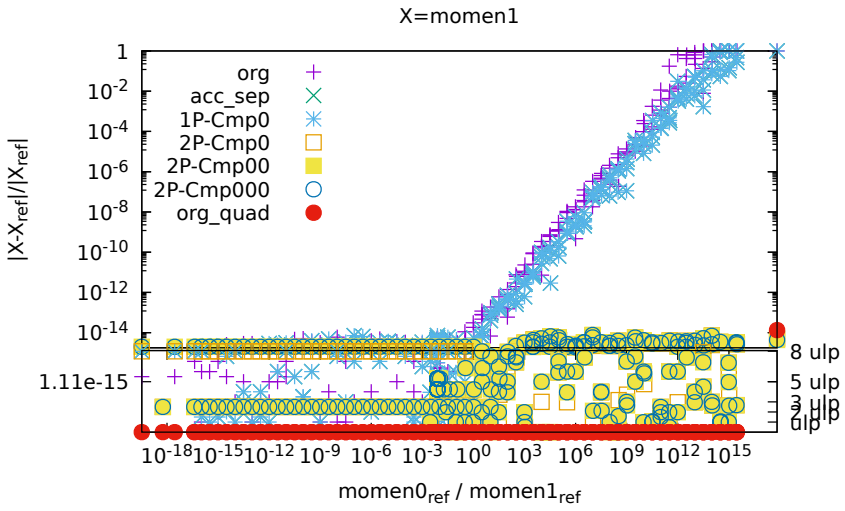




Champagne! Mais est-ce bien nécessaire?

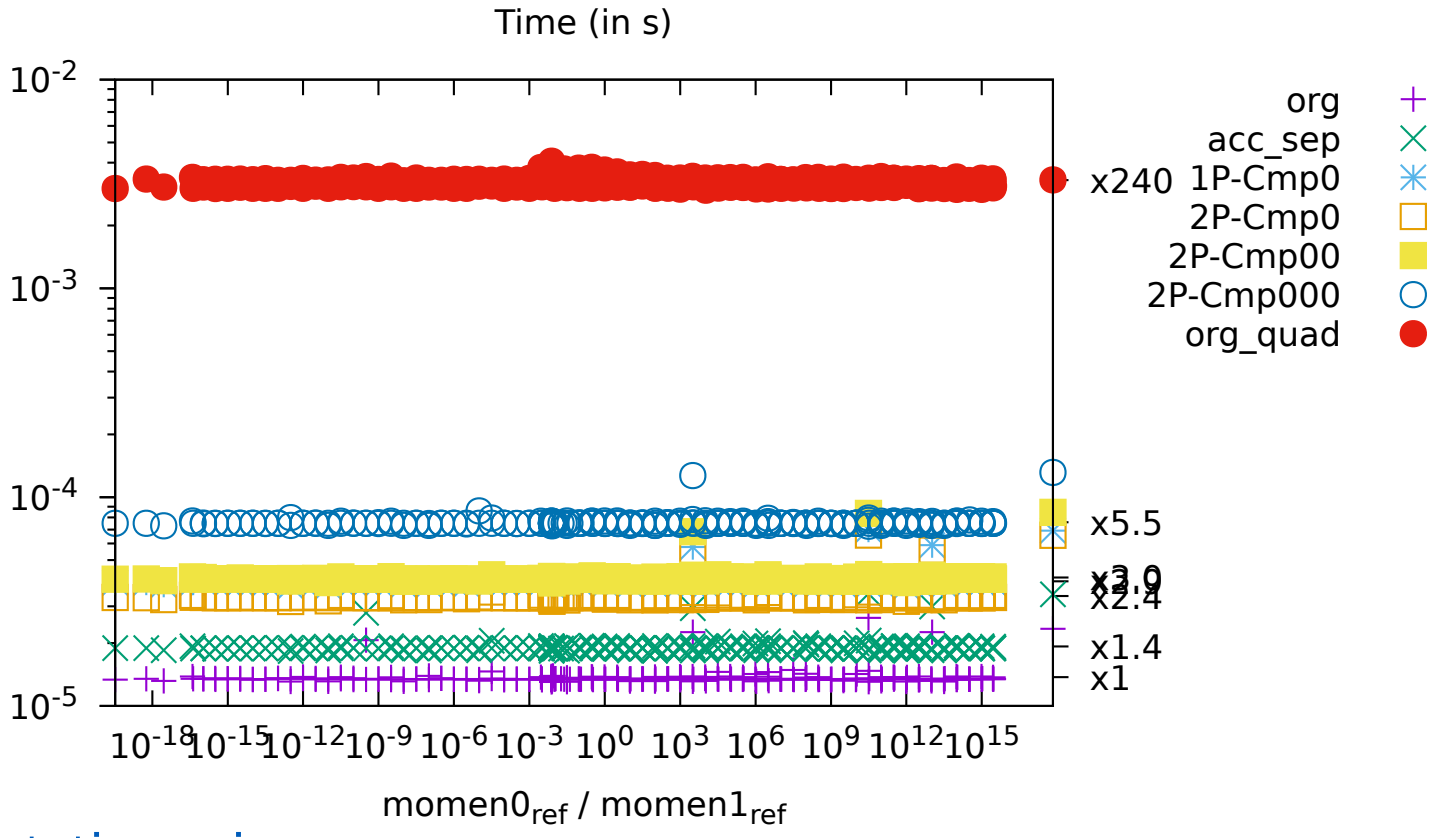
rc32my.F90 => CPP [Org en quad]





- Non vérifiable par Verrou
- Attention à la performance

Les temps



Implémentation naive :

- pas de parallélisme mémoire partagée ;
- pas de vectorisation ;
- pas d'optimisation twoSum/fastTwoSum ;

Difficulté d'intégration

libeft : `https://github.com/ffevotte/libeft`

- Pour des sommes / produits scalaires de données alignées en mémoire, il serait possible de faire des bibliothèques avec des interfaces proches de BLAS. Pour le reste, on doit coder soit même les algorithmes sur base d'une bibliothèque EFT.
- Il faut faire attention aux optimisations du compilateur (Suppression des EFT ou remplacement des fma). libeft passe par les instrinsics x86.
- Performance twoProd : besoin des fma hardware : `-mfma`
- Performance : Attention à l'inlining!
  - En C++ pas de problème (bibliothèque header only)
  - Pour le fortran, il est nécessaire d'utiliser les flags lto :
    `-flto -ffat-lto-objects`

Plan

1. Quelques corrections dans code_aster
   - Corrections par reformulation
   - Corrections par algorithmes compensés
2. REX sur les outils pour la localisation

REX

- Verrou + GDB : c'est lent et limité (pas de watch, pas de reverse...)
  - pour des cas spécifiques : erreur valgrind sur FLT_MAX, Inf et NaN.
- Le delta-debug c'est lent :
  - Début d'implémentation paralléle
  - Récupération de l'information de delta-debug précédent (test des ensembles ddmin précédents avec des heuristiques pour le décalage de lignes, chainage entre verrou_dd_sym et verrou_dd_line)
- Défaut de factorisation du code :
  - mode random_det (sujet à approfondir)
- Les méthodes de couvertures souffrent de beaucoup de faux positifs.
- Lien entre méthodes de localisation et corrections.
  - Les méthodes de couvertures (lignes ou BB) poussent vers les correctifs de type reformulation, modification de tolérance, modification de modèle.
  - Les méthodes de delta-debug poussent vers les approches algorithmes compensés
- L'articulation entre les deux approches est complexe :
  - Mettre en place des algorithmes compensés avant un branchement instable peut compliquer le debug en rendant les cas d'échec moins fréquents.
  - En appliquant la méthode de couverture sur une configuration de delta-debug, on peut réduire le nombre de faux positifs.

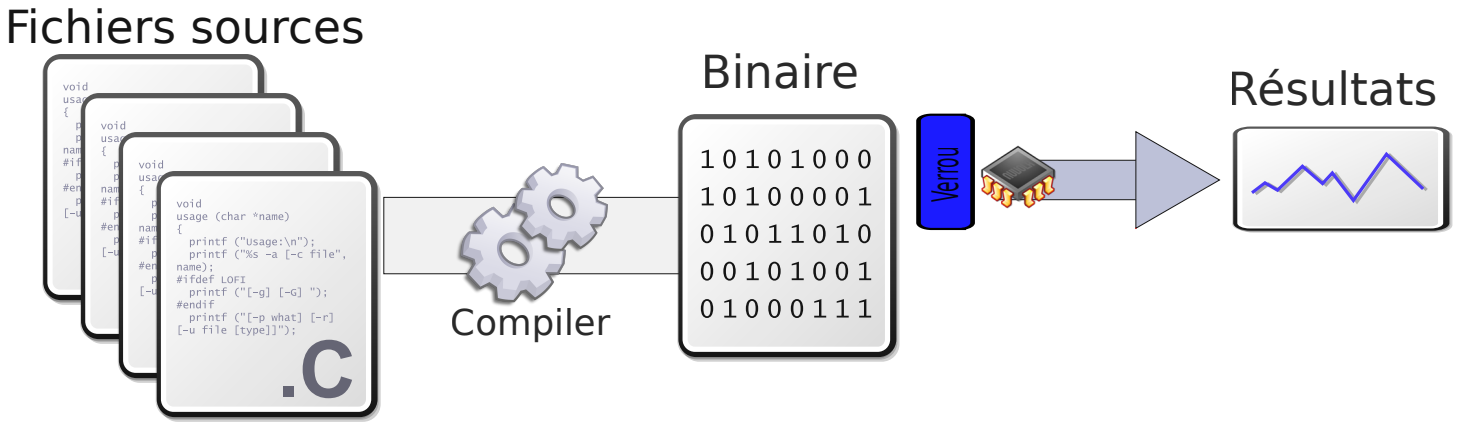
post_verrou_dd

Mise en place d'un outil **post_verrou_dd** qui permet de relancer des calculs sur des configurations obtenues par delta-debug :

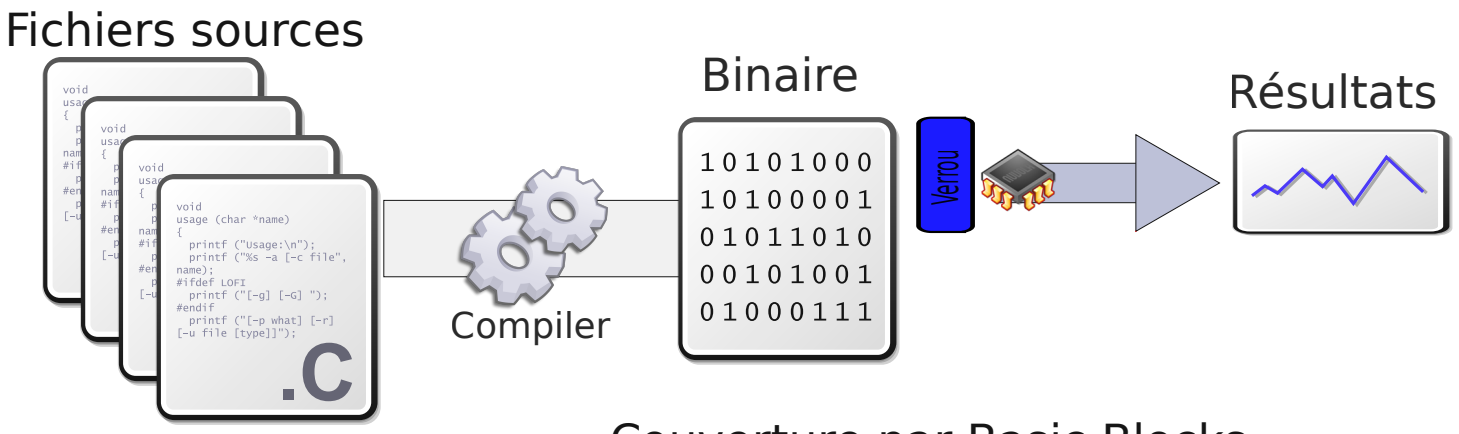
- En ajoutant la couverture par BB (sans effort) ;
- En selectionnant d'autres modes d'arrondi ;
- En sélectionnant des configurations d'instructions (Add,Sub,Mul...).

Remarque : format compatible avec **verrou_plot_stat** (plot les histogrammes).

Méthode de couverture par Basic Blocks



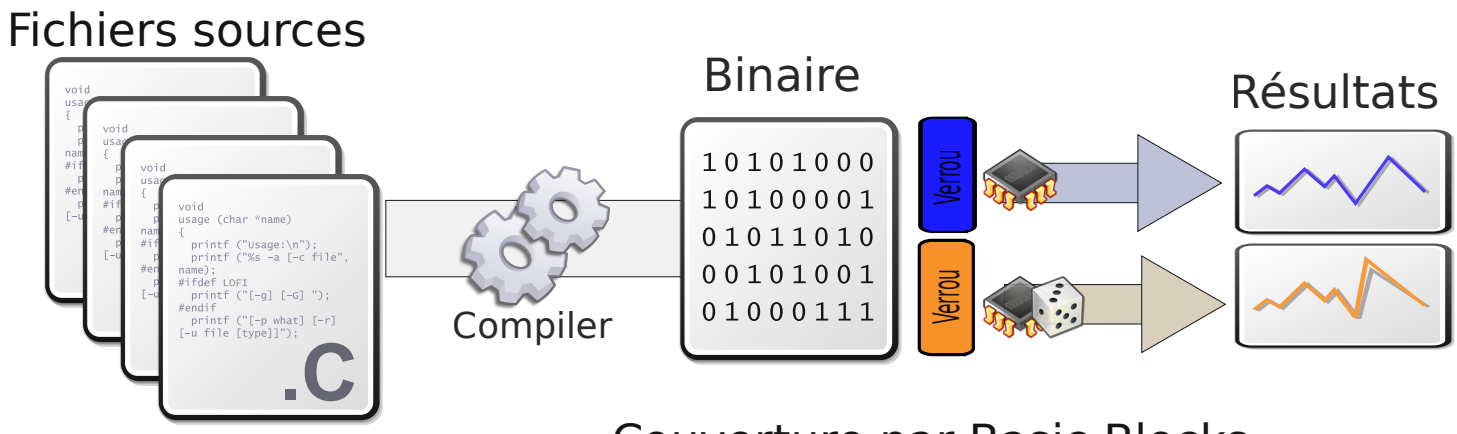
Méthode de couverture par Basic Blocks



Couverture par Basic Blocks

```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032824 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1032935 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Méthode de couverture par Basic Blocks



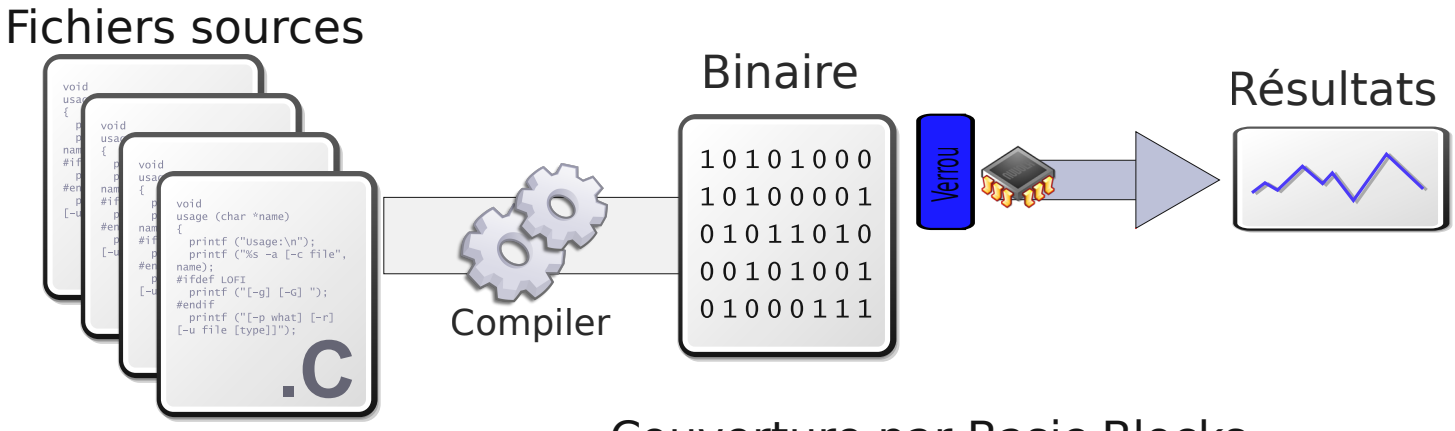
Couverture par Basic Blocks

```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032948 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1033059 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

DIFF / MELD

```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032824 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1032935 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Méthode de couverture par Basic Blocks

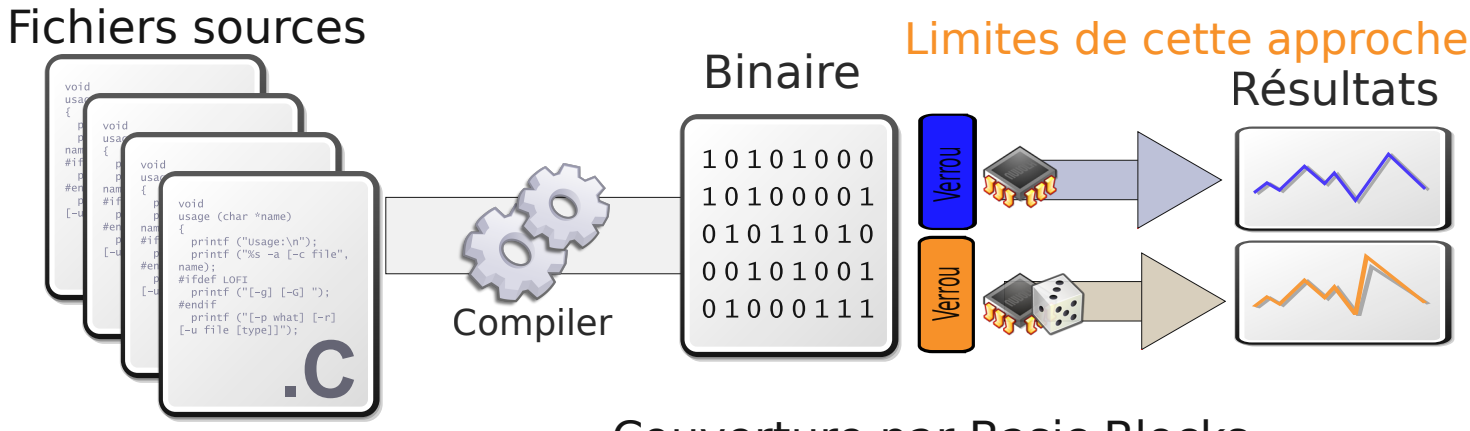


Couverture par Basic Blocks

```
template <typename T>
RealType integrate (const T& f, RealType ↩
      a, RealType b, unsigned int n) {
  // Pas d'intégration
  const RealType dx = (b - a) / n;

  // Accumulateur
  RealType sum = 0.;

  // Boucle sur les rectangles d'intégration
  for (RealType x = a + 0.5 * dx ;
       x < b ;
       x += dx) {
    sum += dx * f(x);
  }

  return sum;
}
```

```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032824 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1032935 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Méthode de couverture par Basic Blocks

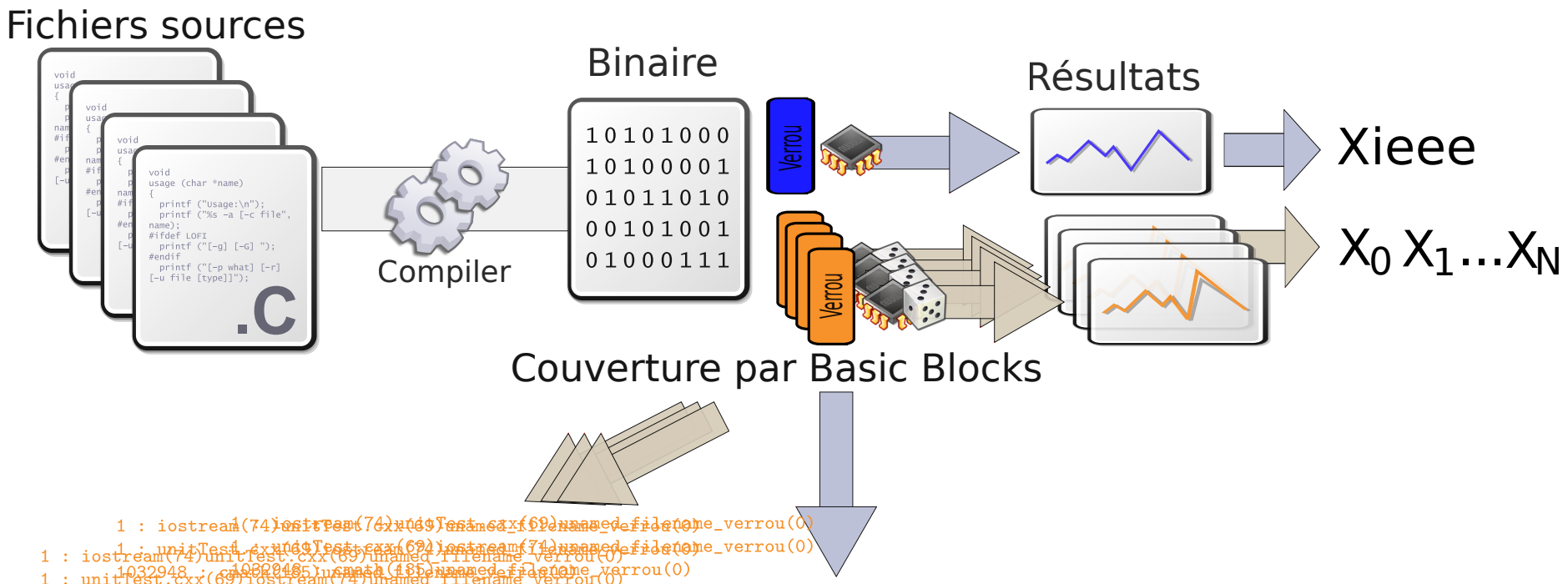



```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032948 : cmath(185)unamed_filename_verrou(0)
112 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
112 : integrate.hxx(21)ios_base.h(726)ostream(196)
1033059 : integrate.hxx(23,20-21)
112 : locale_facets.h(874)
112 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
112 : ostream(228)unamed_filename_verrou(0)
112 : ostream(228)unamed_filename_verrou(0)
112 : ostream(228,591)basic_ios.h(450,49)
112 : ostream(561,196)unamed_filename_verrou(0)
112 : ostream(561,228)unamed_filename_verrou(0)
112 : ostream(613)unamed_filename_verrou(0)
112 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

DIFF / MELD

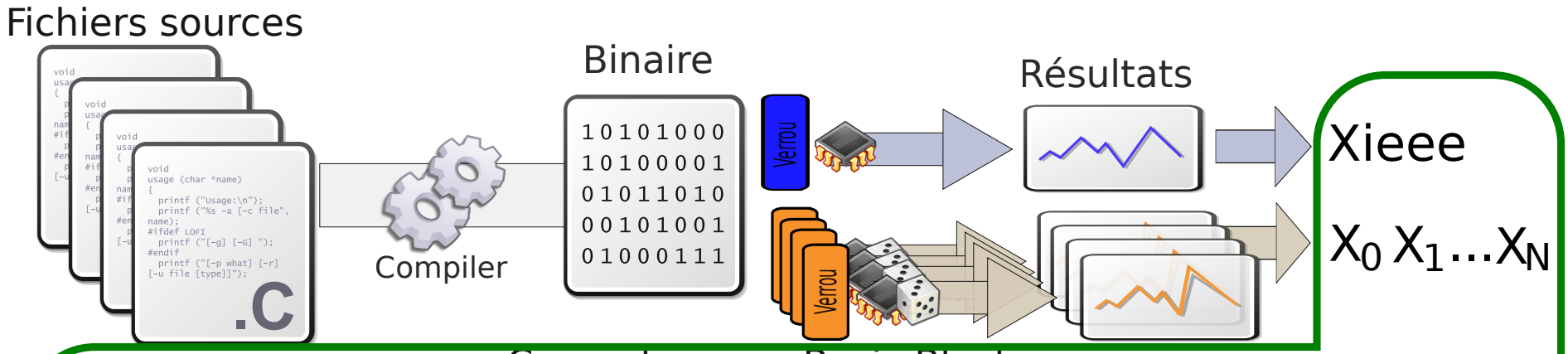
```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032824 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1032935 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Méthode de couverture par Basic Blocks




```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032824 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)ios_base.h(726)ostream(196)
1032935 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Méthode de couverture par Basic Blocks



Couverture par Basic Blocks

```
1 : iostream(74)unitTest.cxx(69)unamed_filename_verrou(0)
1 : unitTest.cxx(69)iostream(74)unamed_filename_verrou(0)
1032948 : cmath(185)unamed_filename_verrou(0)
111 : integrate.hxx(21)cmath(185)unamed_filename_verrou(0)
1033059 : integrate.hxx(23,20-21)
111 : locale_facets.h(874)
111 : locale_facets.h(875)ostream(591)unitTest.cxx(25)
111 : ostream(228)unamed_filename_verrou(0)
111 : ostream(228,591)basic_ios.h(450,49)
111 : ostream(561,196)unamed_filename_verrou(0)
111 : ostream(561,228)unamed_filename_verrou(0)
111 : ostream(613)unamed_filename_verrou(0)
110 : unitTest.cxx(24)integrate.hxx(14,20,17,21)
...
```

Analyse de données

Analyse de données implique jeux de données

Travail avec un collègue EDF : Raphaël Marc.

Difficulté de l'approche

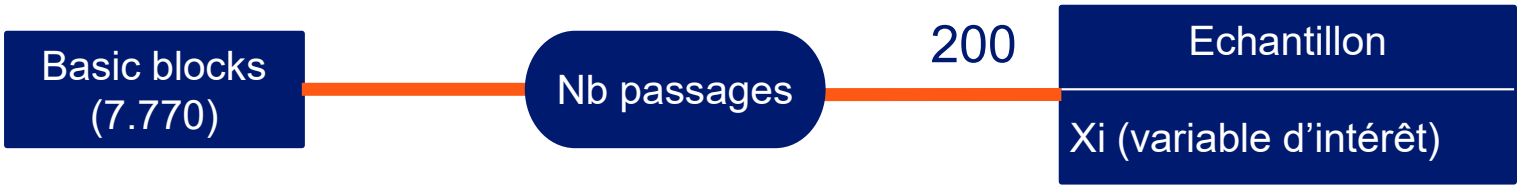
- La réalité terrain ne s'obtient qu'en corrigeant le code ;
- Il n'existe pas de base de donnée liant code+dataset et localisation d'erreur ;
- Sur des cas simples, l'approche n'a pas d'intérêt.

Choix du jeu de données

- codes jouets/ad-hoc : trop simple ;
- **code_aster : rccm01c** ;
  - temps de calcul court ;
  - en parallèle de cette étude, on applique l'ensemble des méthodes existantes.

Analyse de donnée sur le cas test rccm01c

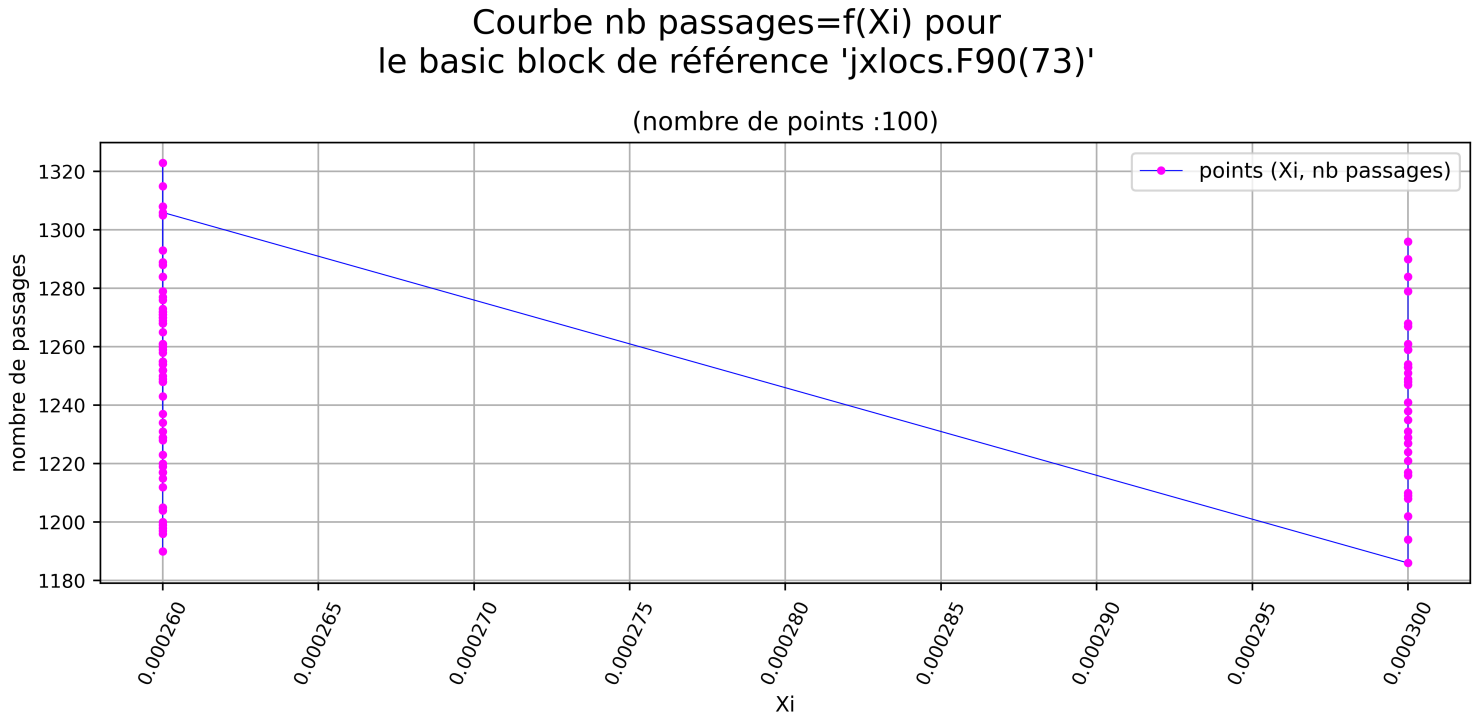
Données



- Pré-traitement : on enlève les basic blocks pour lesquels le *nombre de passages* ne varie pas : $\Rightarrow$ 559 basics blocks ;
- Problème non-supervisé : on ne dispose pas, pour suffisament d'autres cas tests de *labels* (*ie* références de ligne ou de basic blocks provoquant une instabilité par branchement instable).

- Approche : **analyse statistique exploratoire**

Analyse de donnée exploratoire (1/2)

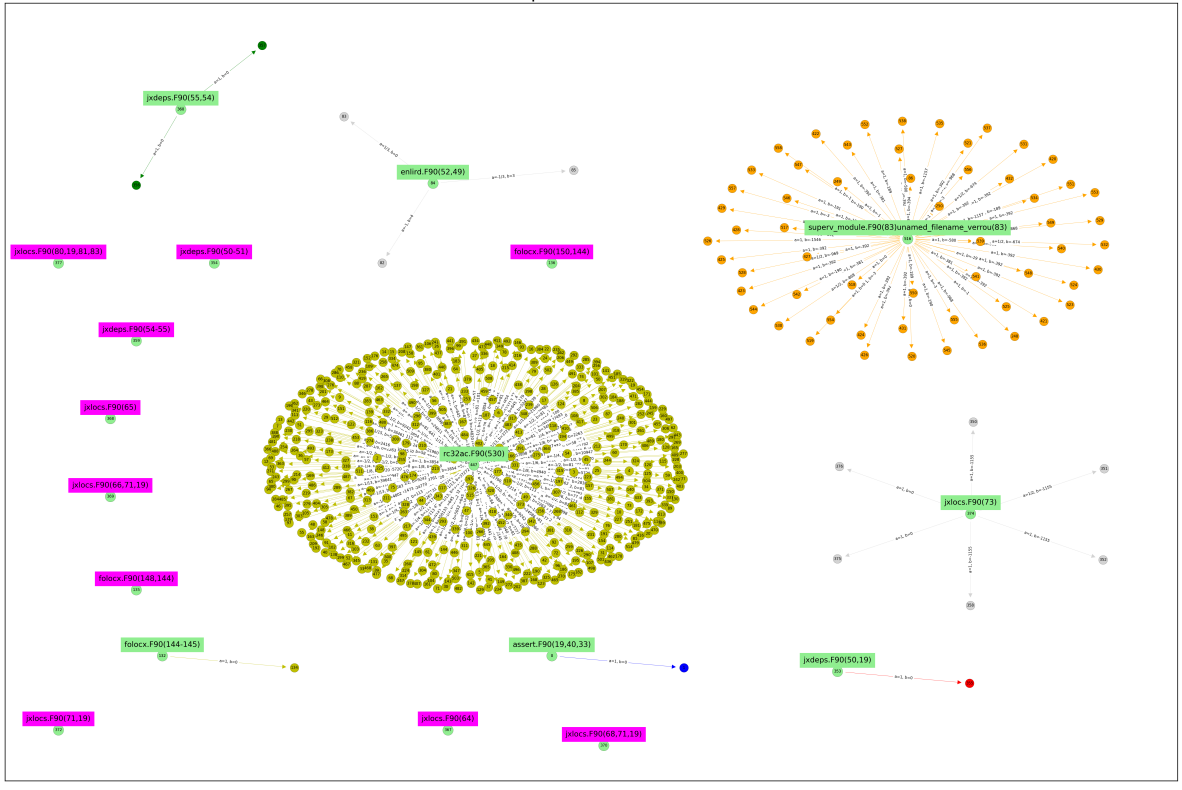


Pas de corrélation entre X_i et le nombre de passages par basic block.

Analyse de donnée exploratoire (2/2)

- On regroupe les 559 courbes (Xi, $nb\_passages$) en classe d'équivalence (existence d'une relation affine à coefficients rationnels).
- Choix d'un représentant (critère : nombre de passages minimal)

Graphe montrant les 559 basic blocks
et la proportionnalité de leurs courbes nb_passages=f(Xi)
559 basic blocks
8 composantes connexes



Perspectives autour de la localisation et Verrou

Frontend Verrou

1. Analyse de donnée : générer avec Verrou les données supplémentaires :
   - informations sur les comparaisons de flottants ;
   - informations temporelles (ordre d'execution) avec des filtres (pour garder des volumes de donnée acceptables).
2. Delta-debug temporel non-intrusif, et chaînable avec les autres Delta-debug (symboles, lignes) et post-traitement.
3. Mettre au propre le delta-debug sur les lignes de python.

Backend Verrou

- Regarder en détail random_det et average_det.
- Intégrer le backend_verrou dans verificarlo pour bénéficier d'un vrai debugger.
- Intégrer le backend_verrou dans verificarlo en version shadow :
  - plusieurs variantes : selon si le mode ieee est dans la shadow ou pas et si on conserve un mode ieee.
  - intérêt : avoir un debugger avec un maximum d'informations
  - risque : les estimations d'erreurs peuvent être mauvaises à cause des problèmes des méthodes synchrones pour les fix-up