Verrou :

L'arithmétique stochastique sans recompiler

RAIM, 7-9 avril 2015

François Févotte
Bruno Lathuilière

EDF R&D
Département SINETICS, Clamart

Sommaire

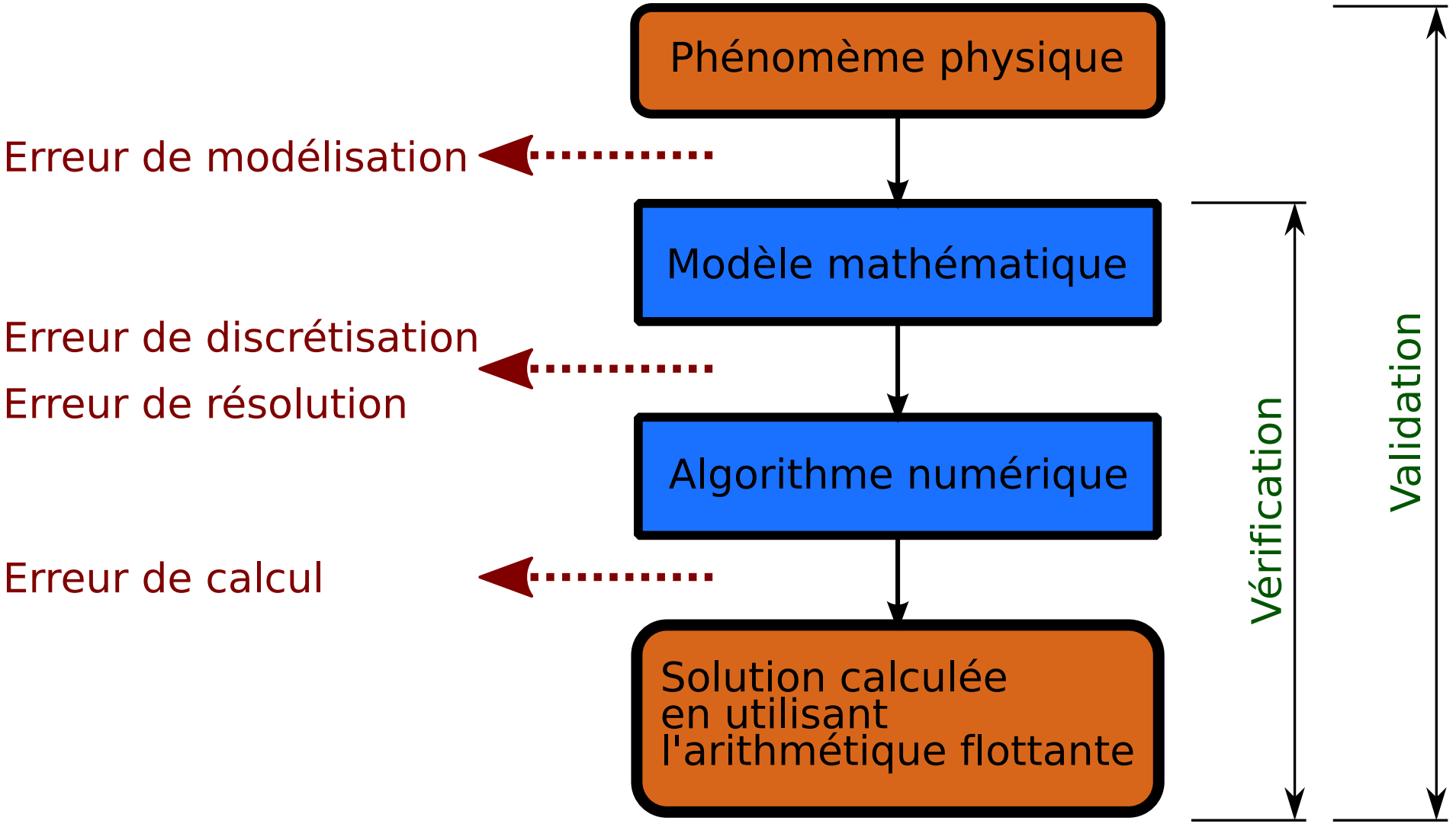
1. Contexte industriel
2. Verrou
3. Mise en pratique : CND
4. Conclusions & perspectives

Contexte industriel

1. Contexte industriel
   - Vérification & Validation
   - Cas d'étude
2. Verrou
3. Mise en pratique : CND
4. Conclusions & perspectives

Contexte

Vérification & validation



Contexte

D'où on part

Codes sous assurance qualité :

- Gestion de versions
- Cas-tests de vérification (et de validation)
- Outils d'analyse de la non-régression

Méthodes de détection artisanales / fortuites :

- Etude de convergence
- Non-reproductibilité lors d'un changement d'infrastructure :
  - compilateur, machine, architecture matérielle
- Non-reproductibilité lors d'une évolution logicielle :
  - optimisation du code, précision (simple/double)
- Non-reproductibilité lors d'un changement de conditions du calcul :
  - nombre de processeurs, re-numérotation du maillage

Contexte

Nos objectifs

Distinguer erreurs de discrétisation / de calcul :

- Améliorer la reproductibilité des résultats :
  - comprendre l'origine des erreurs d'arithmétique
  - prédire l'importance des écarts
- Corriger / réduire les erreurs de calcul
- Améliorer la performance
  - éviter le calcul inutile / néfaste (itérations sur du bruit)
  - payer le coût de la précision (uniquement) là où il faut

Cas d'étude

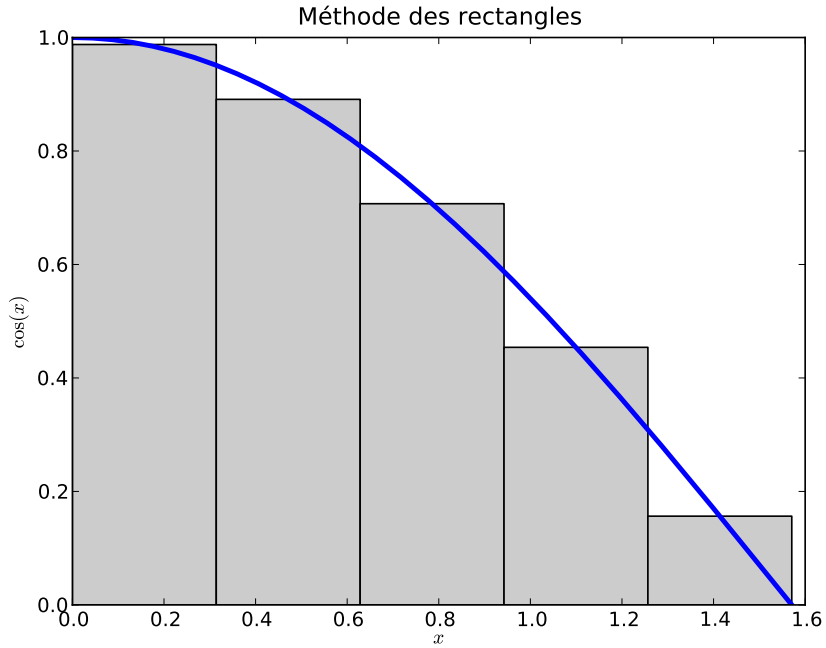
Intégration de *cos*

- On cherche à calculer :

$$I = \int_0^{\frac{\pi}{2}} \cos(x)\, dx.$$

- Référence analytique :

$$I = 1.$$



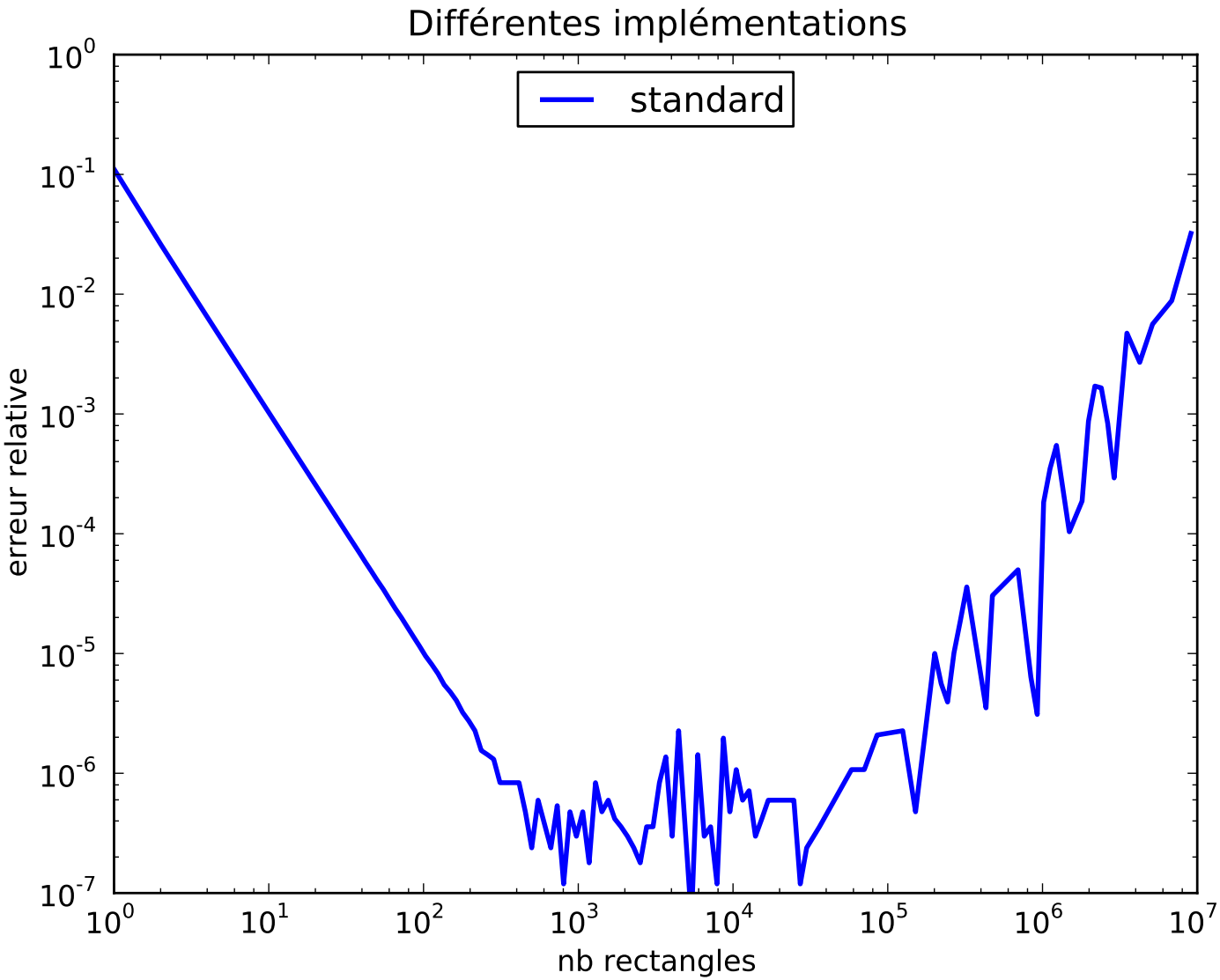
- Calcul approché (méthode des rectangles) :

$$\tilde{I}_n = \sum_{i=1}^{n} \cos(x_i)\, \delta x, \qquad \delta x = \frac{\pi}{2\,n},$$

$$x_i = (i + \frac{1}{2})\delta x.$$

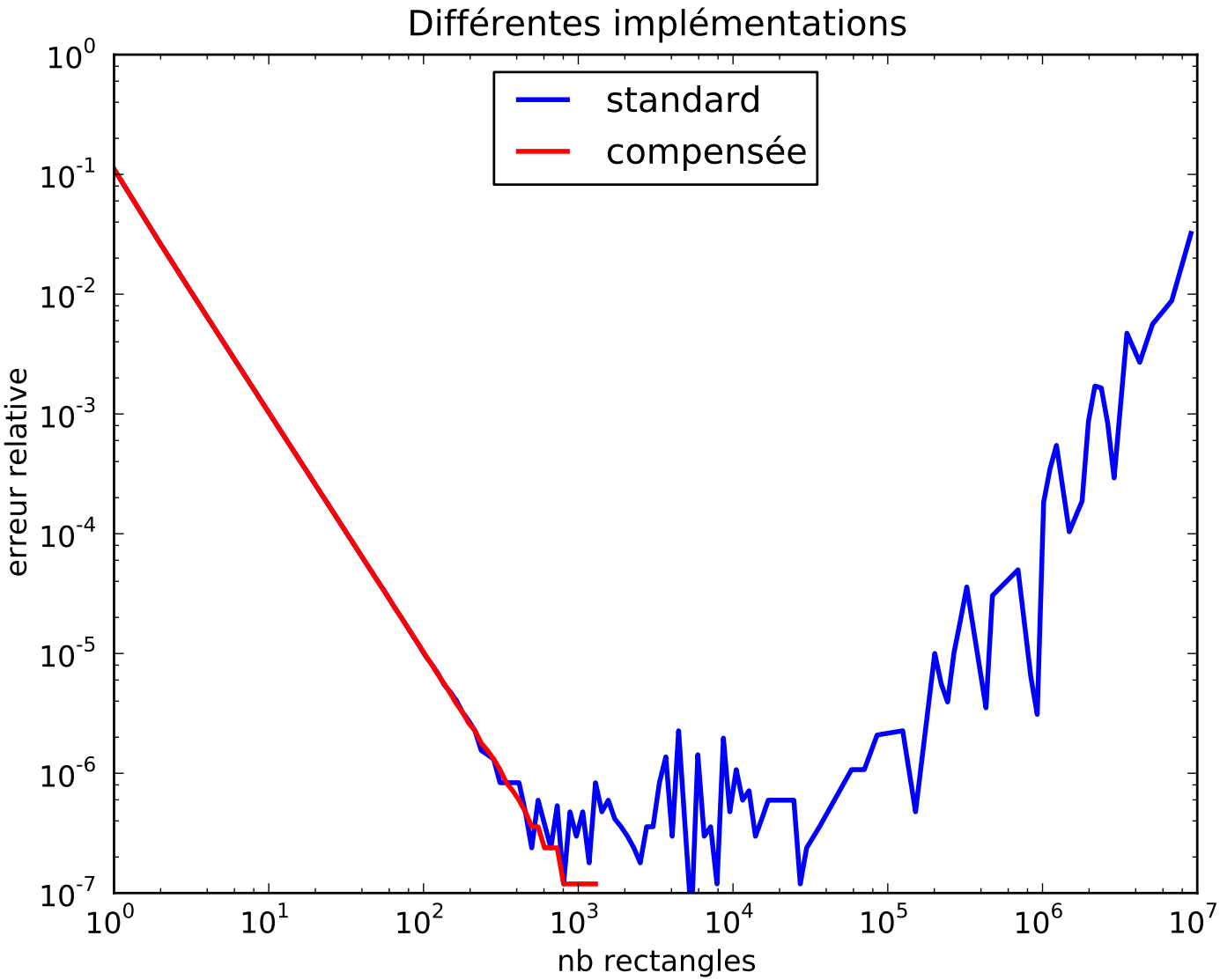
Comparaison des implémentations

Implémentation standard



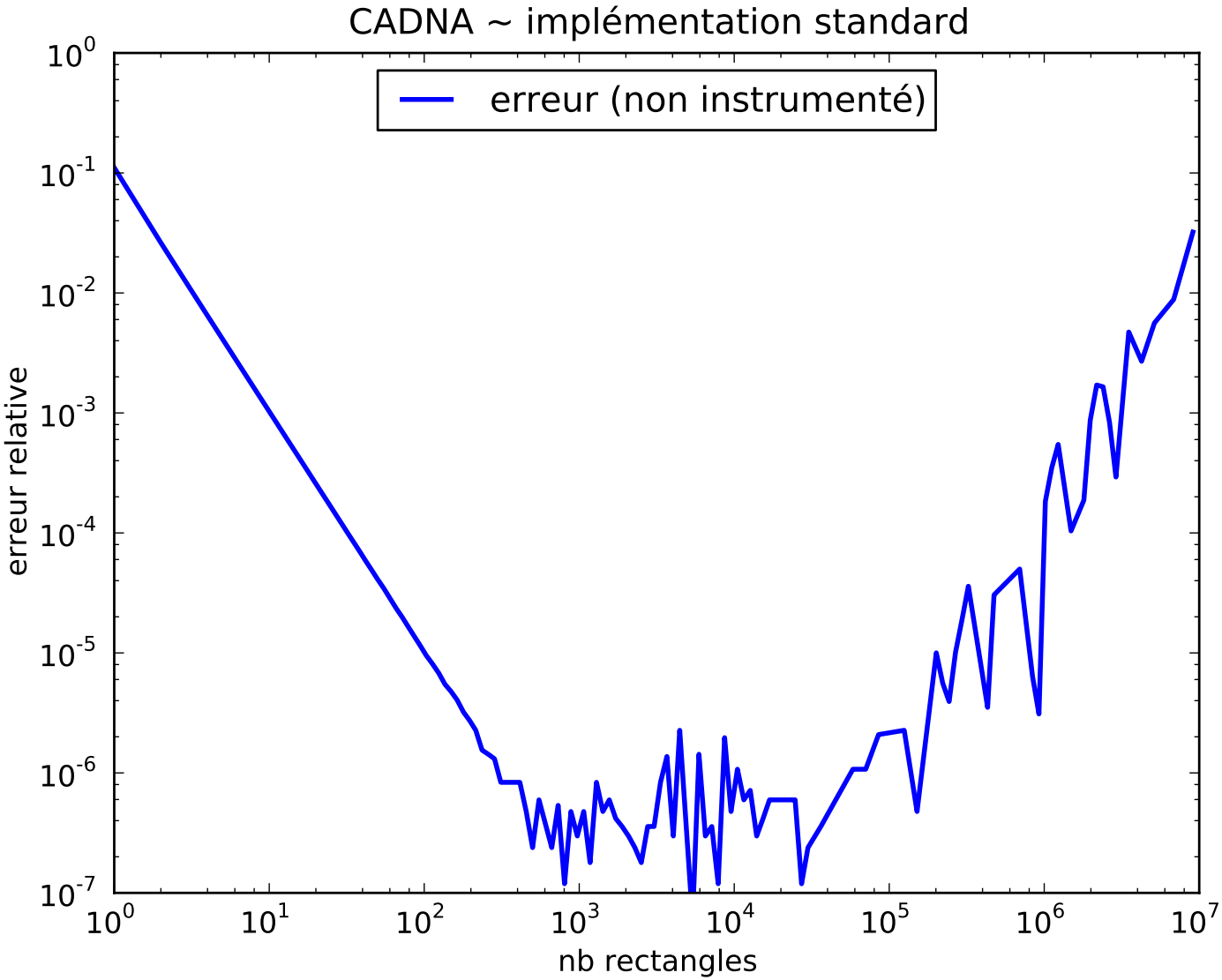
Comparaison des implémentations

Implémentation compensée



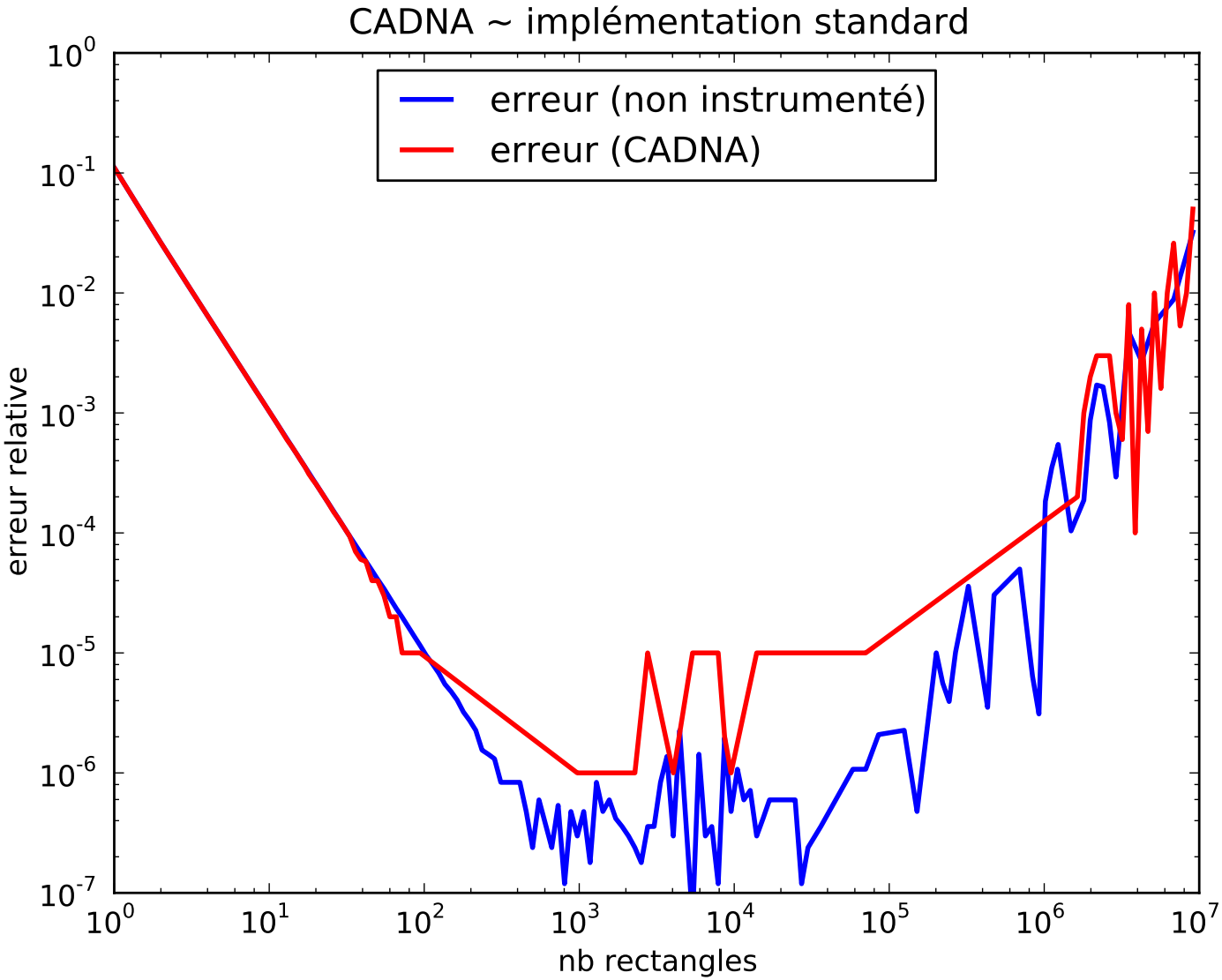
Évaluation avec CADNA

Version standard



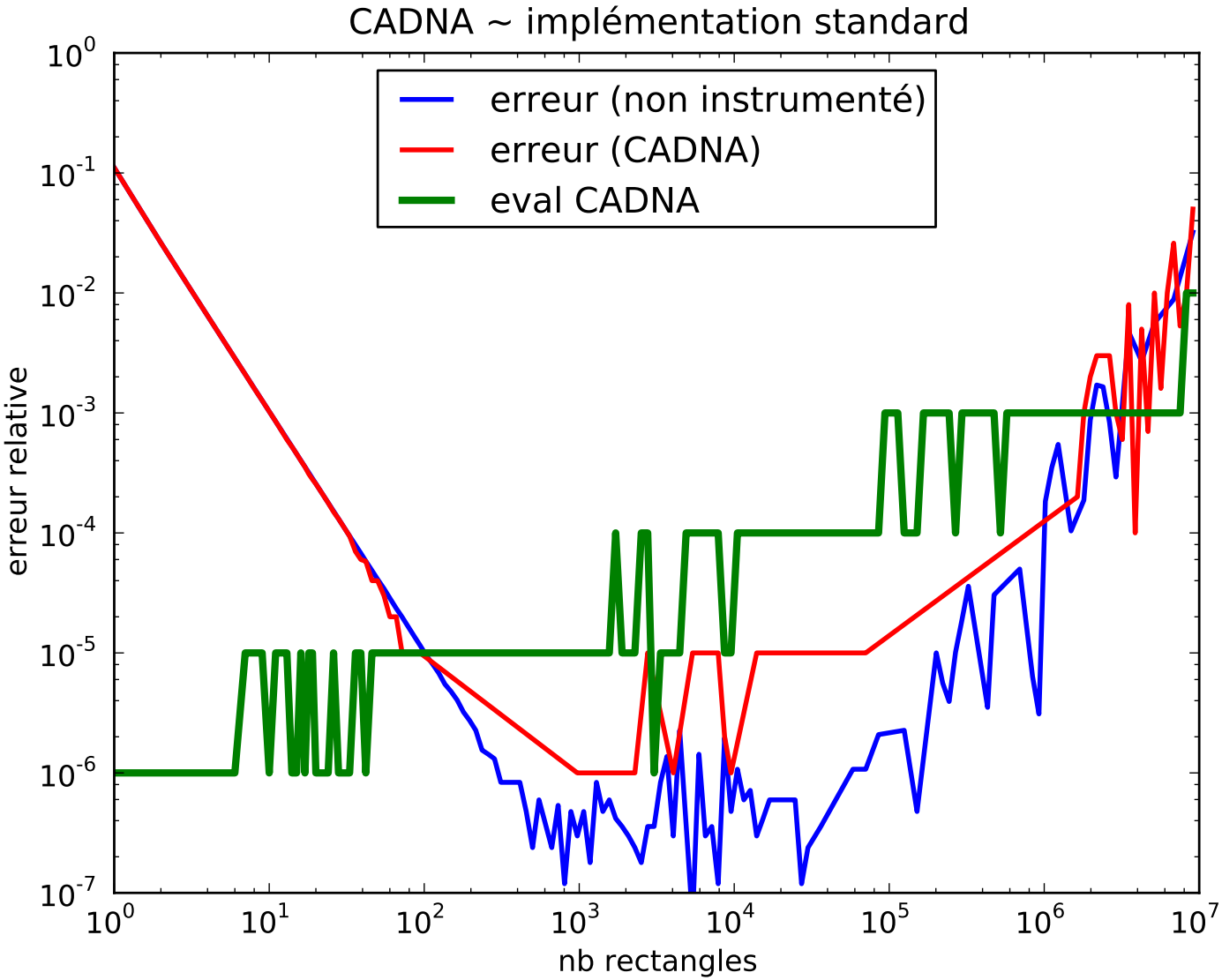
Évaluation avec CADNA

Version standard



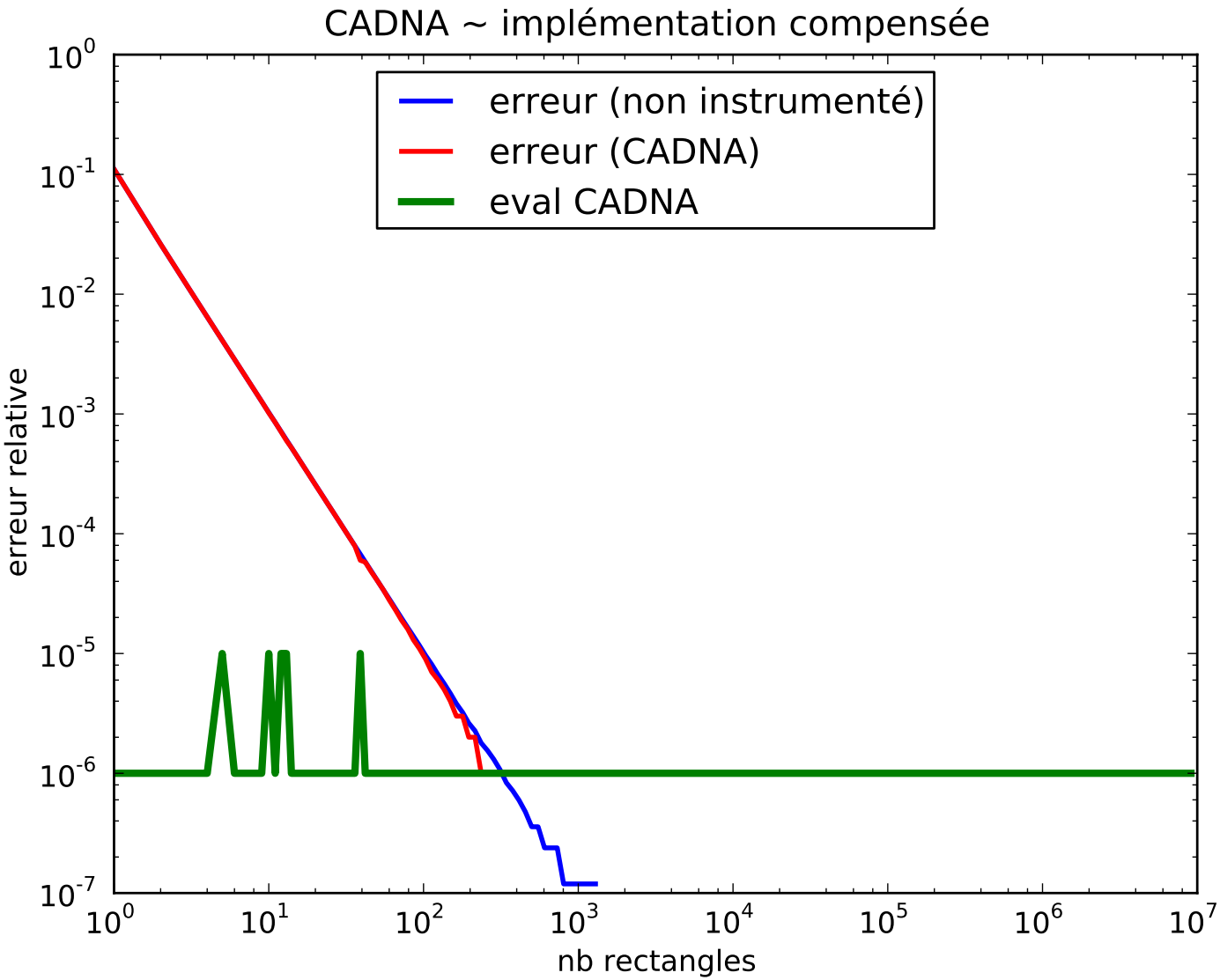
Évaluation avec CADNA

Version standard



Évaluation avec CADNA

Version compensée



Évaluation avec CADNA

Retour d'expérience

Points positifs

- Affichage du nombre de chiffres significatifs prenant en compte la propagation des erreurs
- Détection des instabilités de branchements

Nécessité d'instrumenter le code source

- sur un code *bien écrit** c'est pas très compliqué à faire

  * et selon le type de branchements présents
- sur un code dont on n'a pas l'ensemble des sources c'est impossible !
  - bibliothèques externes
  - produits achetés sur étagère
  - produits co-développés

Évaluation avec CADNA

Retour d'expérience

Points positifs

- Affichage du nombre de chiffres significatifs prenant en compte la propagation des erreurs
- Détection des instabilités de branchements

Nécessité d'instrumenter le code source

- sur un code *bien écrit** c'est pas très compliqué à faire

  * et selon le type de branchements présents
- sur un code dont on n'a pas l'ensemble des sources c'est impossible !
  - bibliothèques externes
  - produits achetés sur étagère
  - produits co-développés

⇒ Notre objectif : étudier l'instrumentation dynamique du binaire

Verrou

1. Contexte industriel

2. Verrou
   - Instrumentation des opérations
   - Évaluation de la précision

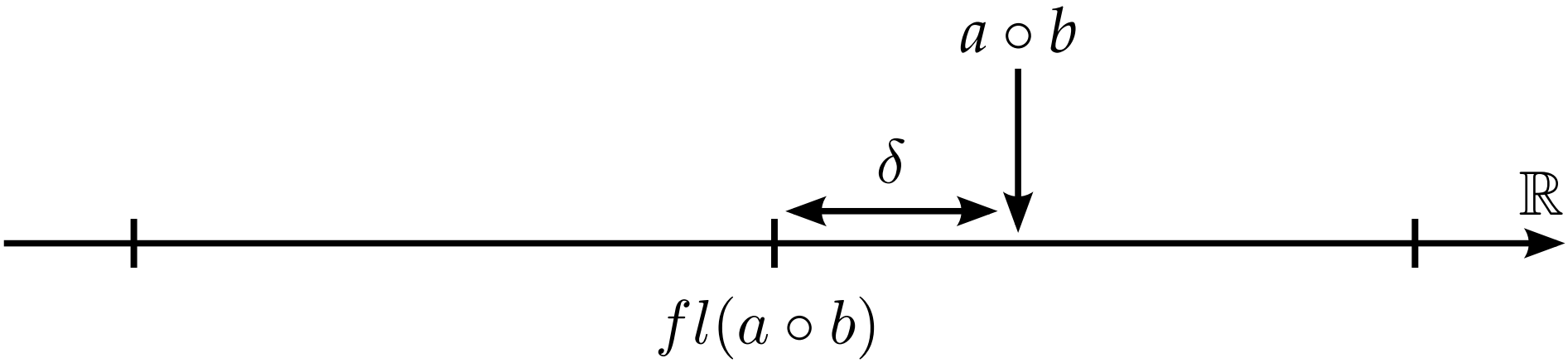
3. Mise en pratique : CND

4. Conclusions & perspectives

Exemple d'utilisation de verrou

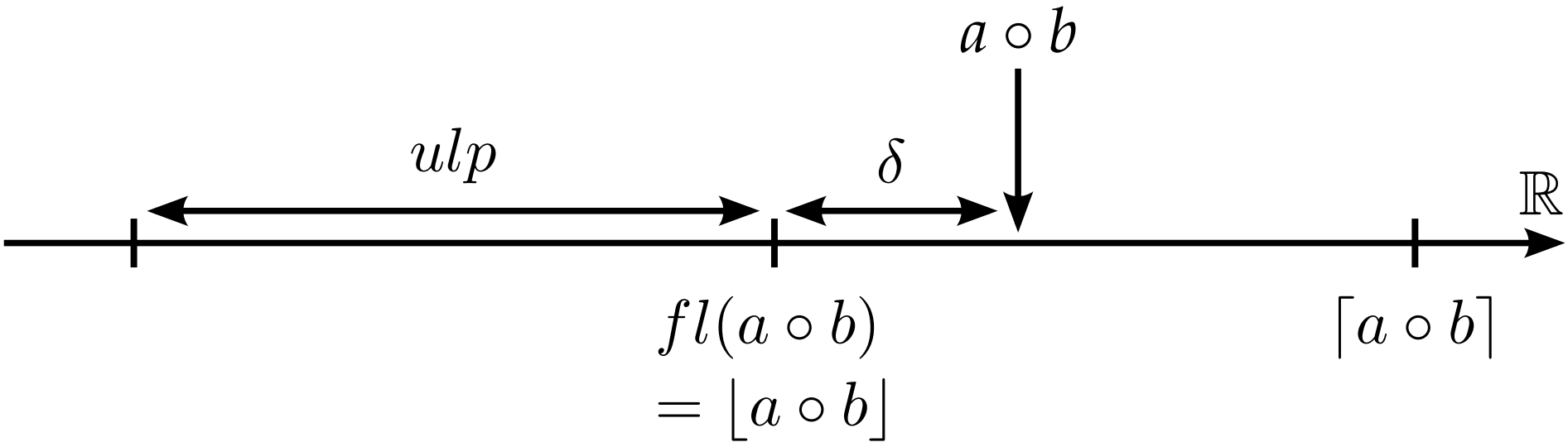
```
$ valgrind --tool=verrou --rounding-mode=random BINAIRE [ARGS...]

==16147== Verrou, Check floating-point rounding errors
==16147== Copyright (C) 2014, F. Fevotte & B. Lathuiliere.
...
==16147== First seed : 1429023813
==16147== Simulating AVERAGE rounding mode
==16147== Instrumented operations :
==16147==   add : yes
...
==16147==     ----------------------------------------
==16147==     Operation
==16147==      `- Precision
==16147==     ----------------------------------------
==16147==     add                  500869335
==16147==      `- flt                  400695468
==16147==      `- dbl                  100173867
==16147==     ----------------------------------------
==16147==     sub                  763127658
==16147==      `- flt                  763127658
==16147==     ----------------------------------------
==16147==     mul                 1202086563
==16147==      `- flt                 1101912537
==16147==      `- dbl                  100174026
==16147==     ----------------------------------------
...
```

Simulation des modes d'arrondi



- Transformation sans erreur :
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$

Simulation des modes d'arrondi



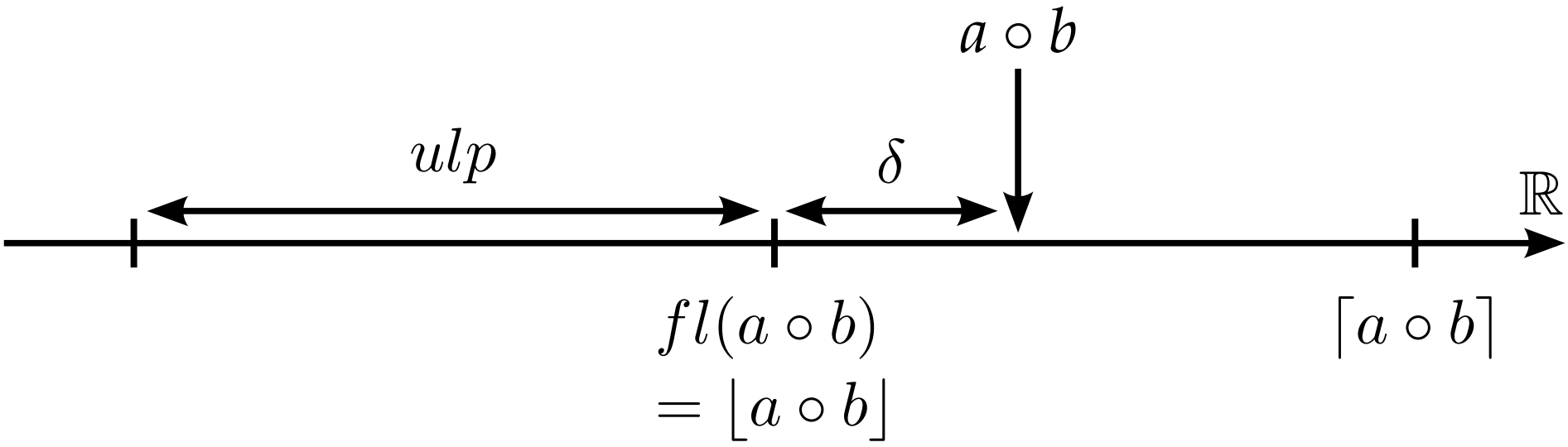
- Transformation sans erreur :
  - $a \circ b = \sigma + \delta$,
  - $\sigma = fl(a \circ b)$
- Si $\delta > 0$:
  - $\lfloor a \circ b \rfloor = fl(a \circ b)$,
  - $\lceil a \circ b \rceil = fl(a \circ b) + ulp$.

Simulation des modes d'arrondi

Modes de fonctionnement possibles de verrou :

- mode "fixe" : toujours le même mode d'arrondi
  - `NEAREST`
  - `UPWARD`
  - `DOWNWARD`
  - `TOWARD_ZERO`
- mode "aléatoire" : un des 2 modes d'arrondi (`UPWARD`, `DOWNWARD`) choisi au hasard pour chaque opération ; équiprobabilité.
- mode "moyenne" : choix aléatoire, mais probabilité dépendant du résultat.

Mode “moyenne”



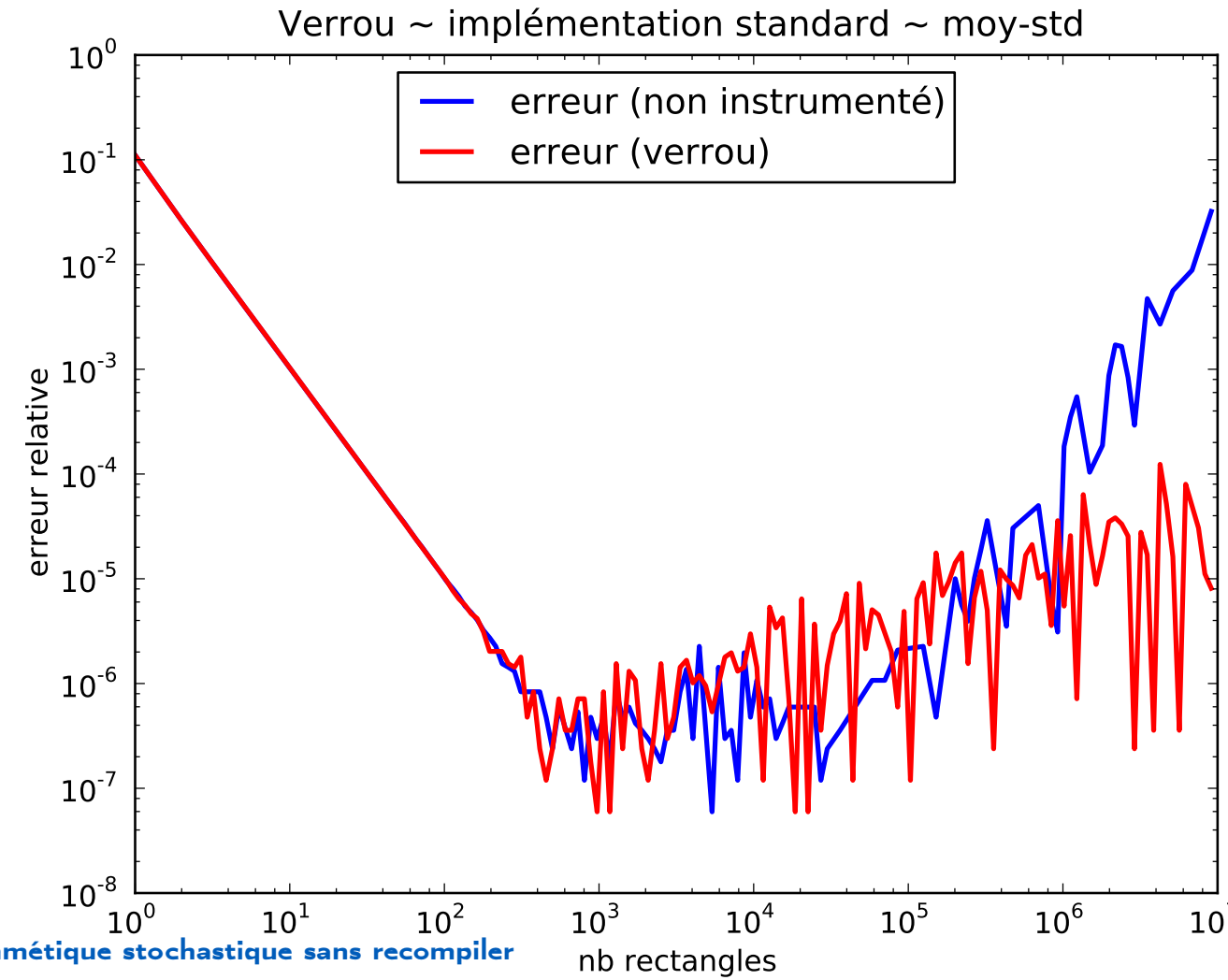
- Tirage aléatoire du résultat r :
  - $p(\lceil a \circ b \rceil) = \dfrac{\delta}{ulp}$
  - $p(\lfloor a \circ b \rfloor) = \dfrac{ulp - \delta}{ulp}$
- “Idée” :
  - $\mathbb{E}(r) = a \circ b$
  - éliminer les biais systématiques en cas d’instructions non indépendantes
  - aucune justification théorique

Évaluation de la précision

Post-traitement

Lancement de plusieurs calculs :

- calcul non instrumenté (arrondi au plus près)
- calcul verrou (mode “moyenne”)

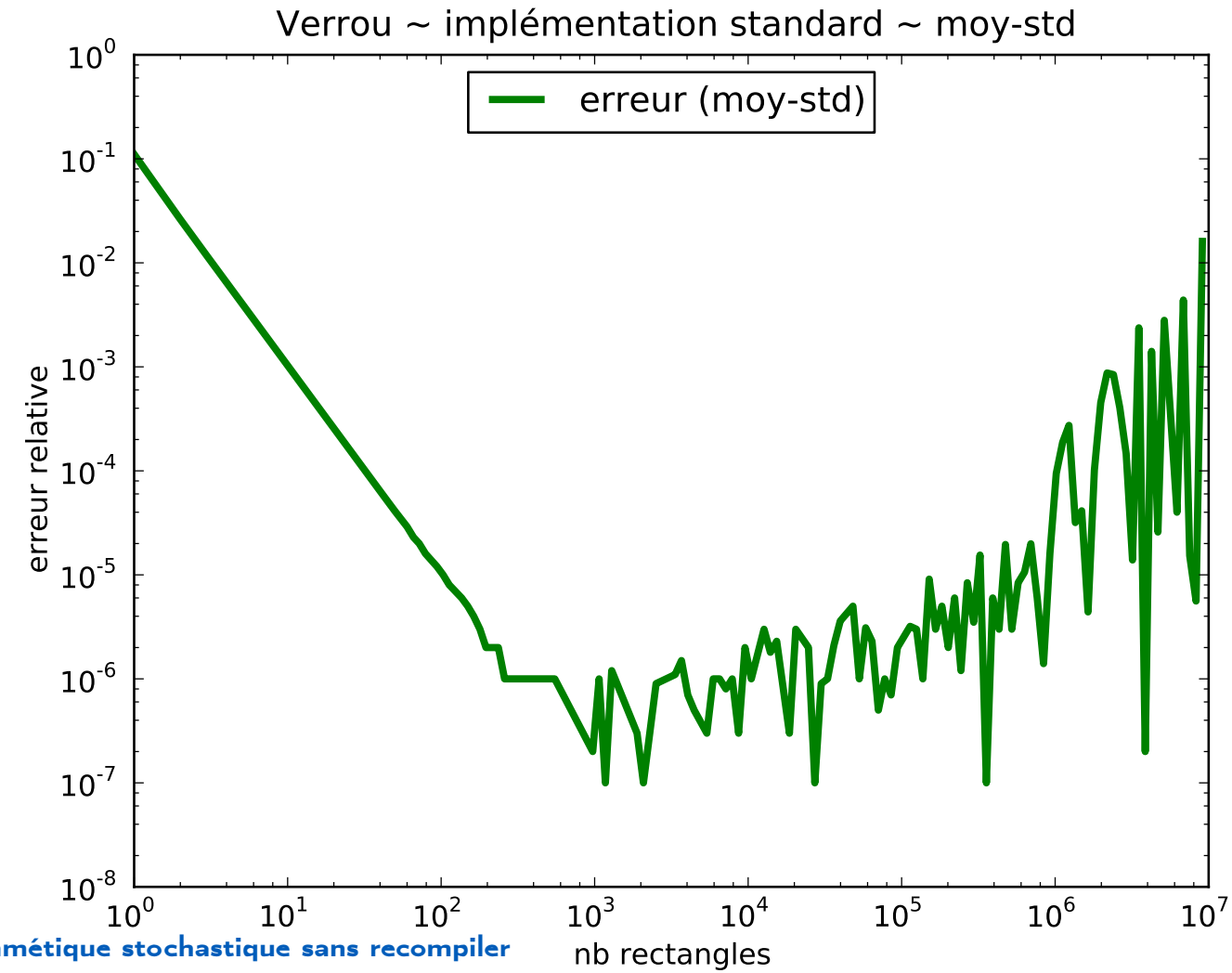


Évaluation de la précision

Post-traitement

- résultat = moyenne

Lancement de plusieurs calculs, puis post-traitement :

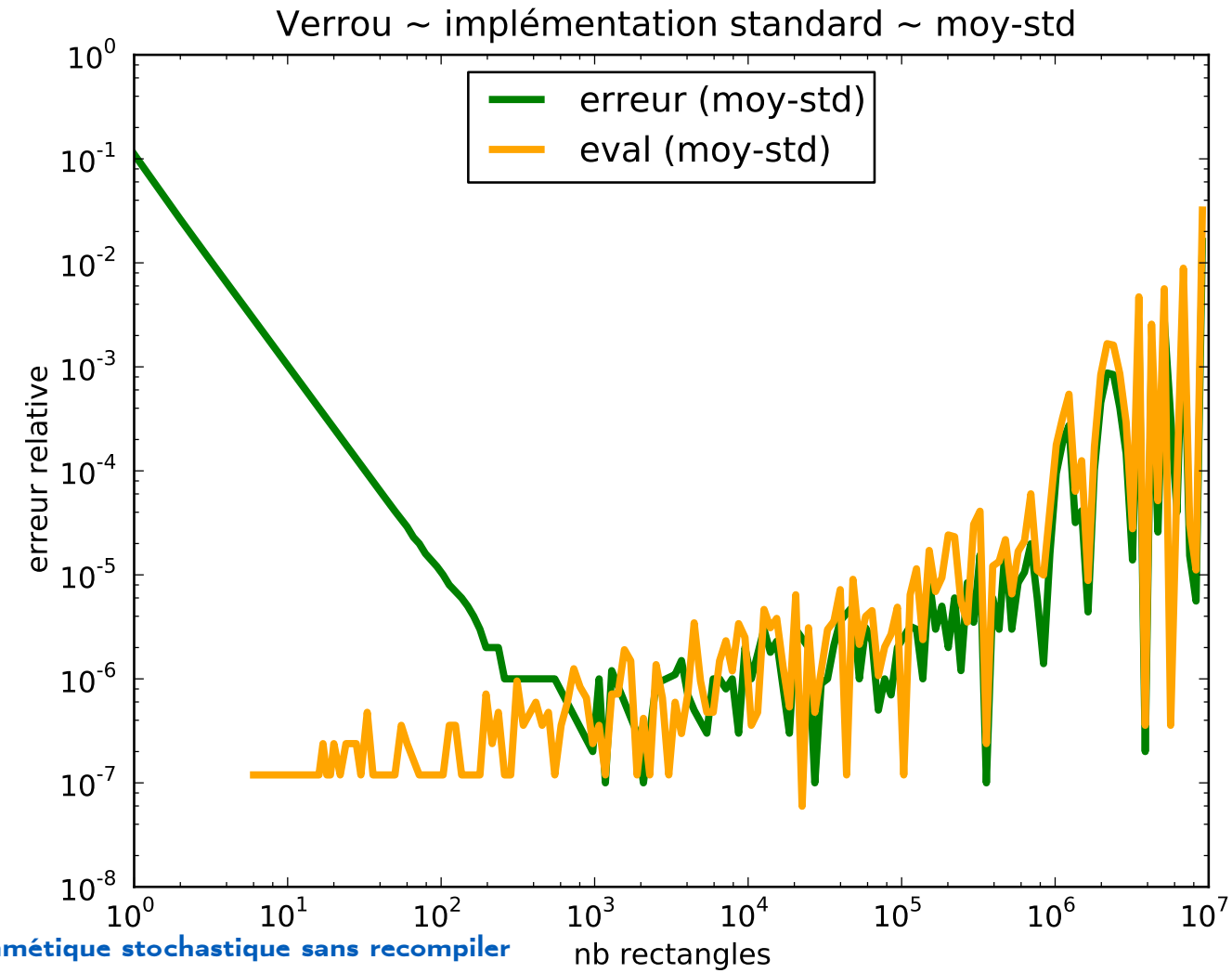


Évaluation de la précision

Post-traitement

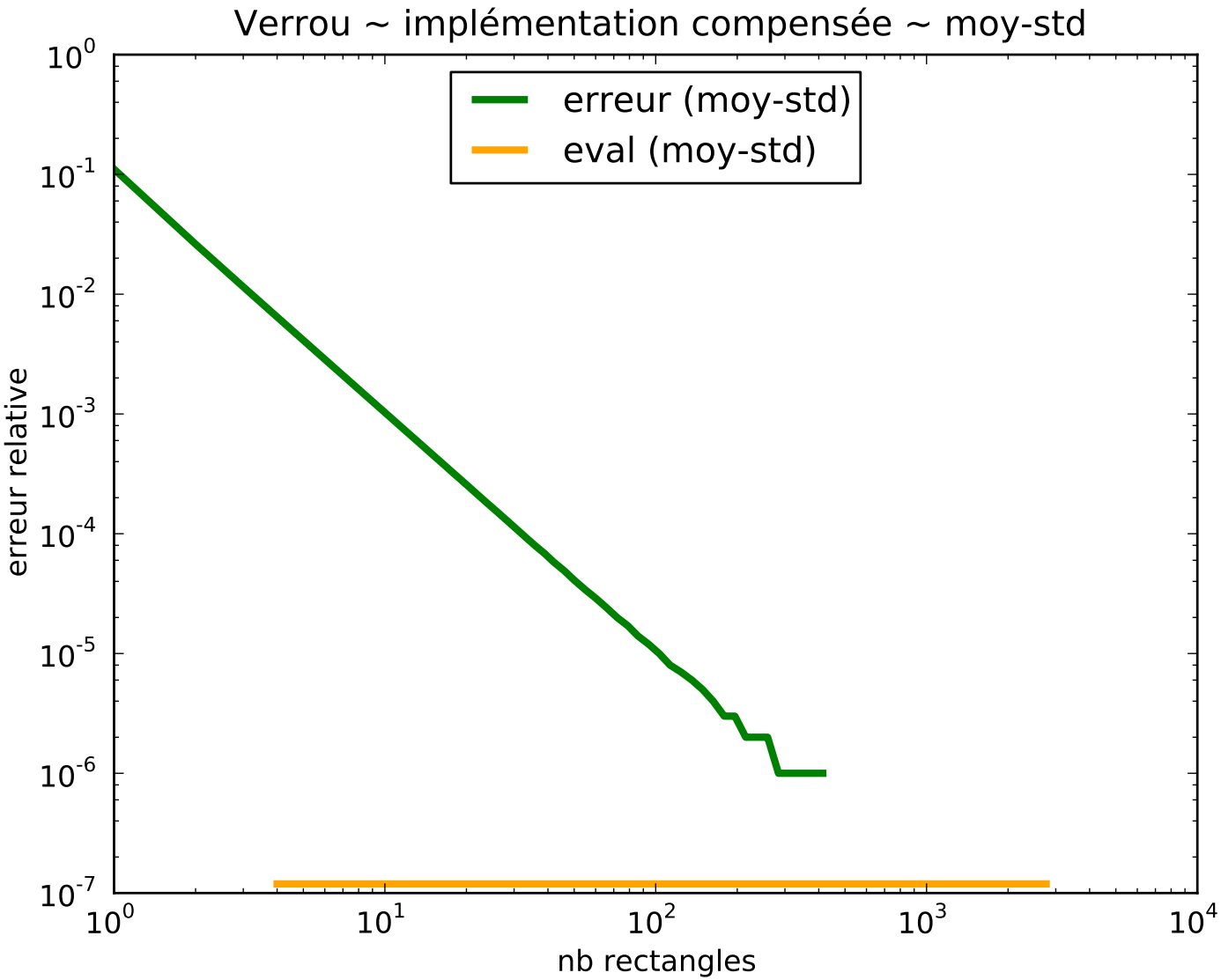
Lancement de plusieurs calculs, puis post-traitement :

- résultat = moyenne
- précision = écart-type



Évaluation de la précision

Méthode “moyenne - standard”



Mise en pratique : CND

1. Contexte industriel

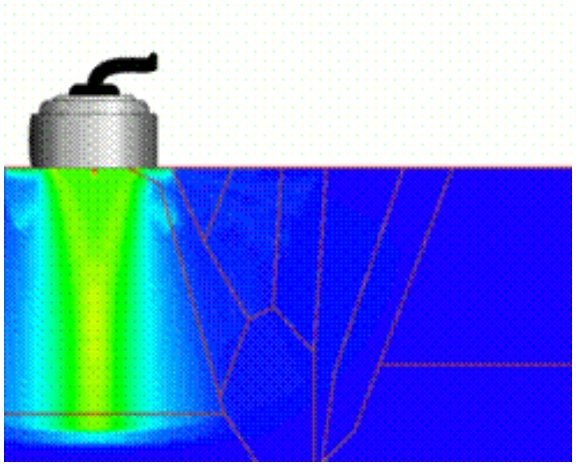
2. Verrou

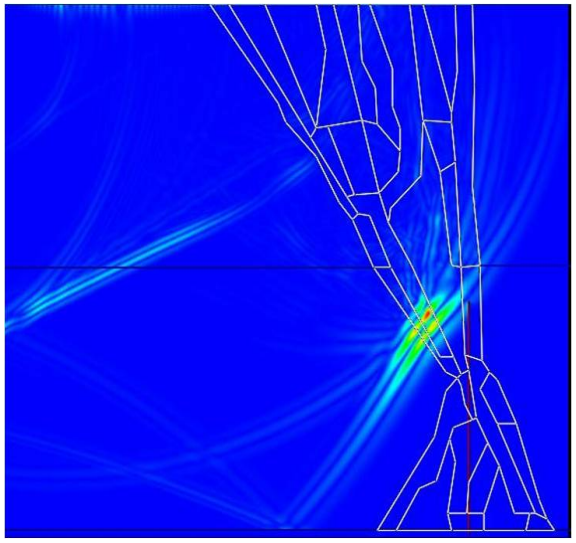
3. Mise en pratique : CND

4. Conclusions & perspectives

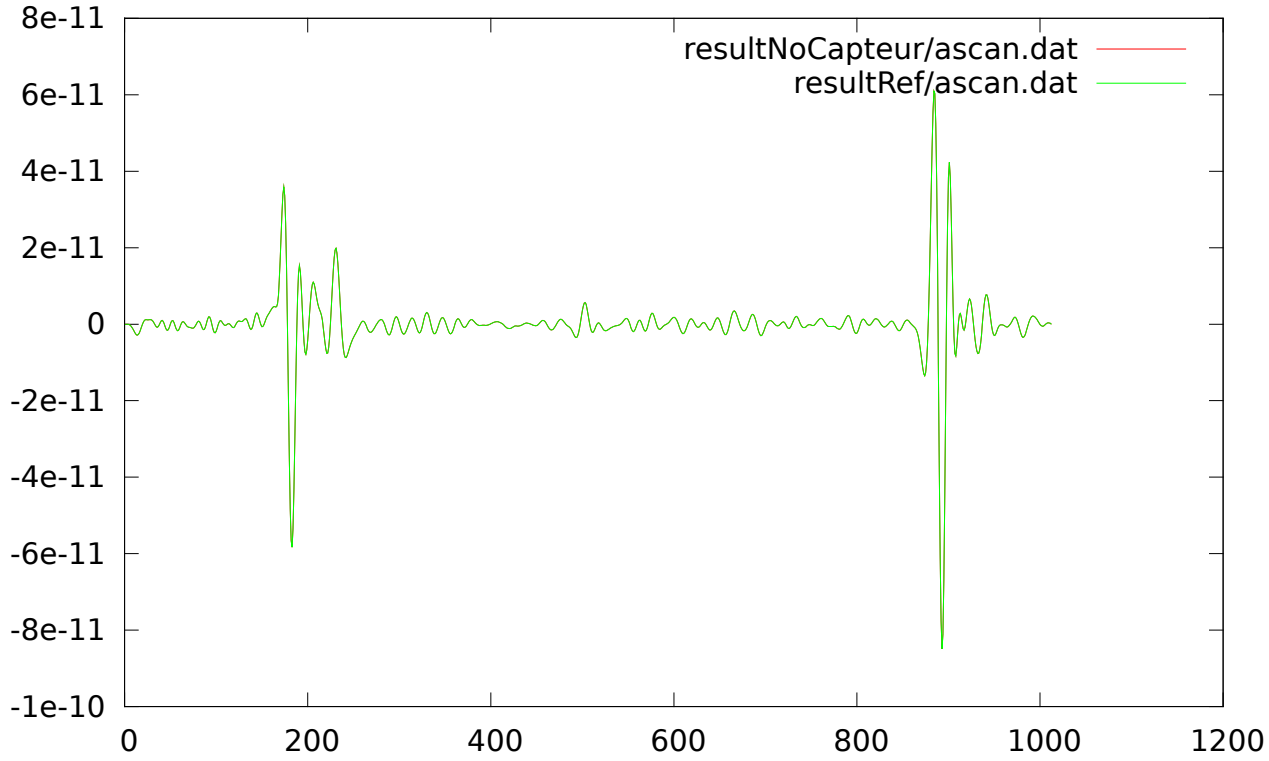
Contrôle non destructif par ultra-sons

Outil de calcul scientifique ATHENA



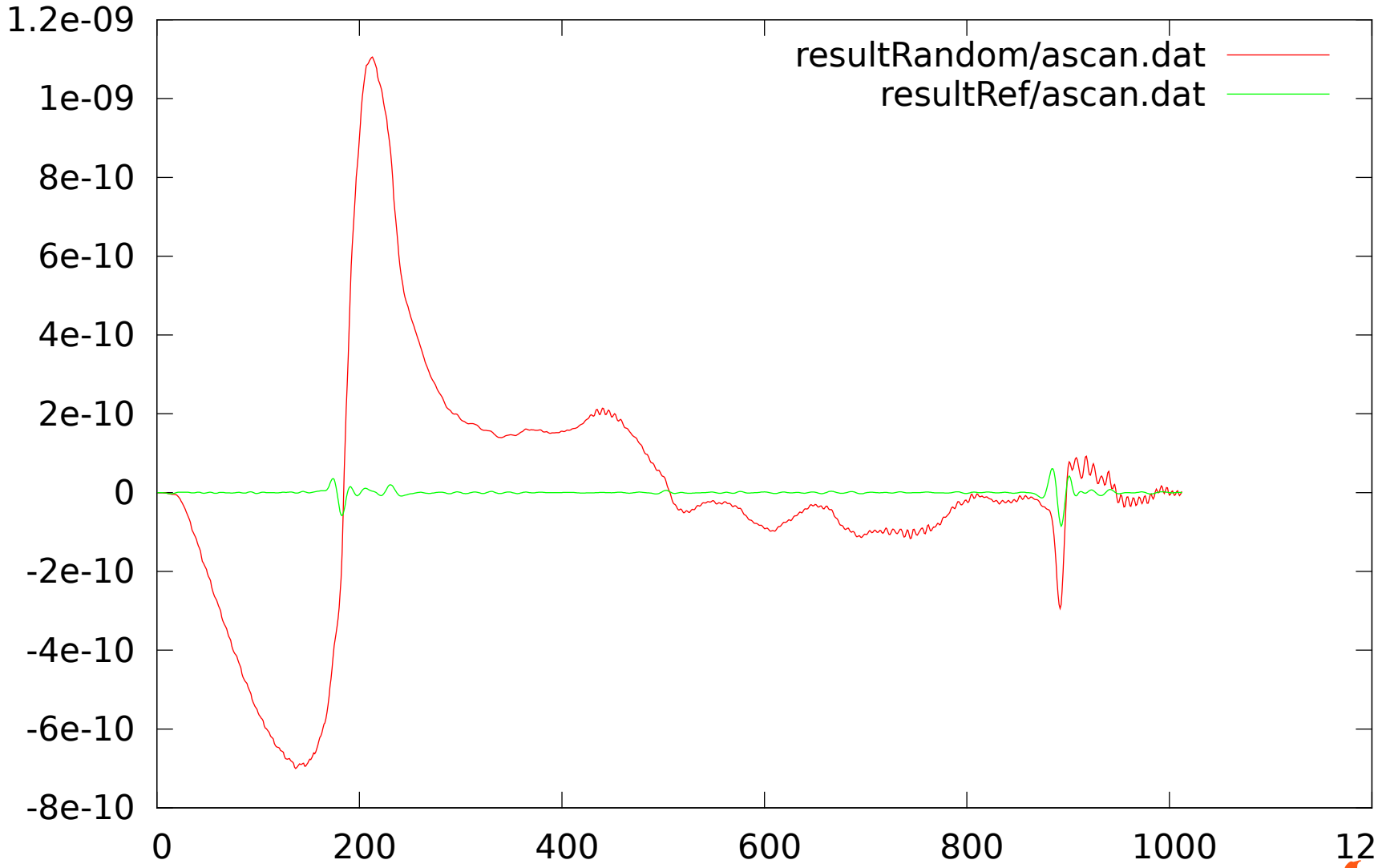


Résultat produit : “A-scan”



Première analyse

Réutilisation des outils de non régression

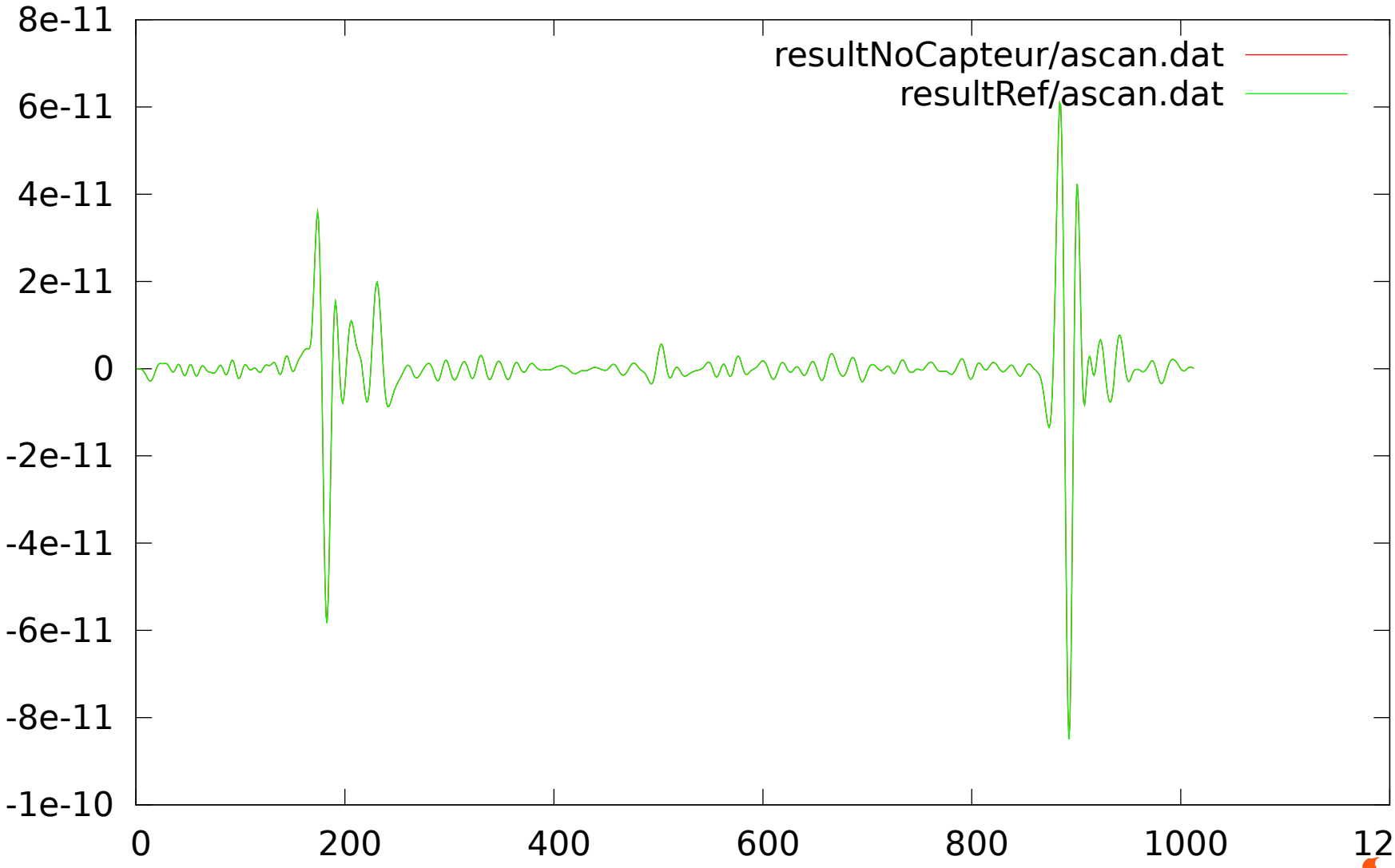


Sections instrumentées

```
1 #include <valgrind/verrou.h>
2 #include <stdio.h>
3
4 float compute ();
5
6 int main () {
7   VERROU_START_INSTRUMENTATION;
8   float result = compute();
9   VERROU_STOP_INSTRUMENTATION;
10
11  fprintf (stdout, "result = %f", result);
12 }
```

Seconde analyse

Tout est instrumenté, sauf la gestion de position du capteur



Sections déterministes

```
1 int posCapteur (float x) {
2   return floor( 0.1 * x );
3 }
4
5 int main () {
6   int maille1 = posCapteur (10.);
7   /* ... */
8   int maille2 = posCapteur (10.);
9   assert (maille1 == maille2);
10 }
```

Sections déterministes

```
1 #include <valgrind/verrou.h>
2
3 int posCapteur (float x) {
4   VERROU_START_DETERMINISTIC(0);
5   const int maille = floor( 0.1 * x );
6   VERROU_STOP_DETERMINISTIC(0);
7   return maille;
8 }
9
10 int main () {
11   int maille1 = posCapteur (10.);
12   /* ... */
13   int maille2 = posCapteur (10.);
14   assert (maille1 == maille2);
15 }
```

Base de non régression

	Réf.	Random1	Random2	Average1	Average2
MultiEltc	1.45s	13.62s (x9)	13.84s (x9)	13.97s (x9)	14.05s (x9)
ins1.dat		0.229818	0.229818	0.000000	0.229818
enerloc.dat		0.112656	0.112656	0.000000	0.112656
sismo.dat		0.191131	0.191131	0.000000	0.191131
ascan.dat		0.220116	0.220116	0.000000	0.220116
CapteurImmersion	0.45s	14.17s (x31)	14.20s (x31)	11.88s (x26)	11.79s (x26)
ins1.dat		0.280809	0.355187	0.105049	0.000000
enerloc.dat		0.234995	0.227002	0.086034	0.000000
sismo.dat		0.160344	0.196354	0.008062	0.000000
ascan.dat		0.290362	0.194622	0.344207	0.016042
don_araignee	10.48s	184.01s (x17)	183.46s (x17)	194.64s (x18)	193.14s (x18)
ins1.dat		0.000000	0.232258	0.000000	0.000000
enerloc.dat		0.000000	0.055204	0.000000	0.000000
ascan.dat		0.000000	0.244644	0.000000	0.000000
ConditionPer...	0.39s	147.77s (x379)	144.15s (x370)	60.65s (x155)	59.93s (x154)

Conclusions & perspectives

1. Contexte industriel

2. Verrou

3. Mise en pratique : CND

4. Conclusions & perspectives

Conclusions & perspectives

Validation numérique des codes

Etude d'Athéna

- Importance pratique des outils de non régression
- Importance des fonctionnalités de l'outil
  - instrumentation partielle (sections)
  - sections déterministes
- Difficile de se passer d'un expert
  - du code (méthodes numériques, interprétation des résultats)
  - de l'arithmétique flottante (compréhension des problèmes)

Vers l'élargissement de ces pratiques

- Tester plus de codes (qu'on connaît)
- Communiquer auprès des équipes de développement
- Communiquer auprès de l'ingénierie

Conclusion & perspectives

Outil verrou

Verrou

- Arithmétique stochastique asynchrone :
  - opérations standard + FMA
  - SSE scalaires + vectorielles (uniquement en double)
- Pas (peu) besoin d'instrumenter les codes
- Temps de calcul : $\times 10 - 30$ en général
  ... mais c'est parfois la loterie ($\times 400$)

Limites et perspectives

- Instrumentation des instructions vectorielles SSE (float) + AVX.
- Fonctions de la bibliothèque mathématique.
- Gestion des optimisations du compilateur (en particulier *reordering*).
- Gestion des instructions scalaires (`-mfpmath=387`) et `long double`.
- Gestion du parallélisme en mémoire partagée.

Merci !

Des questions ?

Annexes

1. Implémentations
2. Objectifs verrou
3. Valgrind
4. Post-traitements Verrou
5. Athena
6. CADNA vs Verrou

Cas d'étude

Implémentation (base)

```
1  float integrate_cos (n)
2  {
3    const float dx = M_PI / (2*n);
4
5    float sum = 0;
6    for ( float x = dx/2 ; x < M_PI/2 ; x += dx )
7      {
8        sum += cos(x) * dx;
9      }
10
11   return sum;
12 }
```

Cas d'étude

Implémentation (améliorée)

```
1  float integrate_cos (n)
2  {
3    const float dx = M_PI / (2*n);
4
5    float sum = 0;
6    for ( int i = 0; i<n; ++i )
7      {
8        const float x = (i+0.5) * dx;
9        sum += cos(x) * dx;
10     }
11
12   return sum;
13 }
```

Cas d'étude

Implémentation (compensée)

```
1  float integrate_cos (n)
2  {
3    const float dx = M_PI / (2*n);
4
5    float sum = 0;
6    float err = 0;
7    for ( int i = 0 ; i<n ; ++i )
8      {
9        const float x = (i+0.5) * dx;
10       (sum, e) = twoSum (sum, cos(x)*dx);
11       err += e;
12     }
13   sum += err;
14
15   return sum;
16 }
```

Objectifs :

se passer de l'instrumentation des sources

1. Instrumentation statique par le compilateur :
   - nécessité d'avoir les sources
   - dépendance au compilateur
   - difficile d'analyser l'impact des options de compilation (`-O3`, `-ffast-math`)
2. Instrumentation dynamique du binaire
   - pas besoin des sources
   - un -g permet souvent une analyse plus fine
   - temps d'exécution plus grand
   - perte d'information (par exemple sur la libmath)

Notre choix : étudier l'instrumentation dynamique du binaire

Instrumentation binaire avec valgrind

- Code C :

```
i++;
y = a * x + b;
```

- Entrée valgrind :

```
i = Add32(i, 1)
t1 = MulF32(a, x)

y = AddF32(t1, b)
```

- Sortie valgrind :

```
i = Add32(i, 1)
Call("count", MUL)
t1 = MulF32(a, x)
Call("count", ADD)
Call("detectCancel", t1, b)
y = Call("myAdd", t1, b)
```

- Possibilités d'instrumentation :
  - Ne rien faire (recopier l'instruction telle quelle)
  - Compter les instructions
  - Détecter des erreurs
  - Remplacer les opérations

Évaluation de la précision

Lancement de plusieurs calculs et post-traitement :

- résultat = moyenne,
- précision = écart type.

Plusieurs stratégies possibles :

- "arithmétique stochastique asynchrone" :
  - N calculs verrou en mode "aléatoire" ;
- "aléatoire - standard" :
  - 1 calcul "standard" non instrumenté (arrondi au plus près),
  - N calculs verrou "aléatoire" ;
- "moyenne - standard" :
  - 1 calcul "standard" non instrumenté (arrondi au plus près),
  - N calculs verrou "moyenne".

Code(s) Athena

Outils de calcul scientifiques

- utilisés par la R&D (2D+3D) + ingénierie (2D)
- sous assurance qualité (non régression)

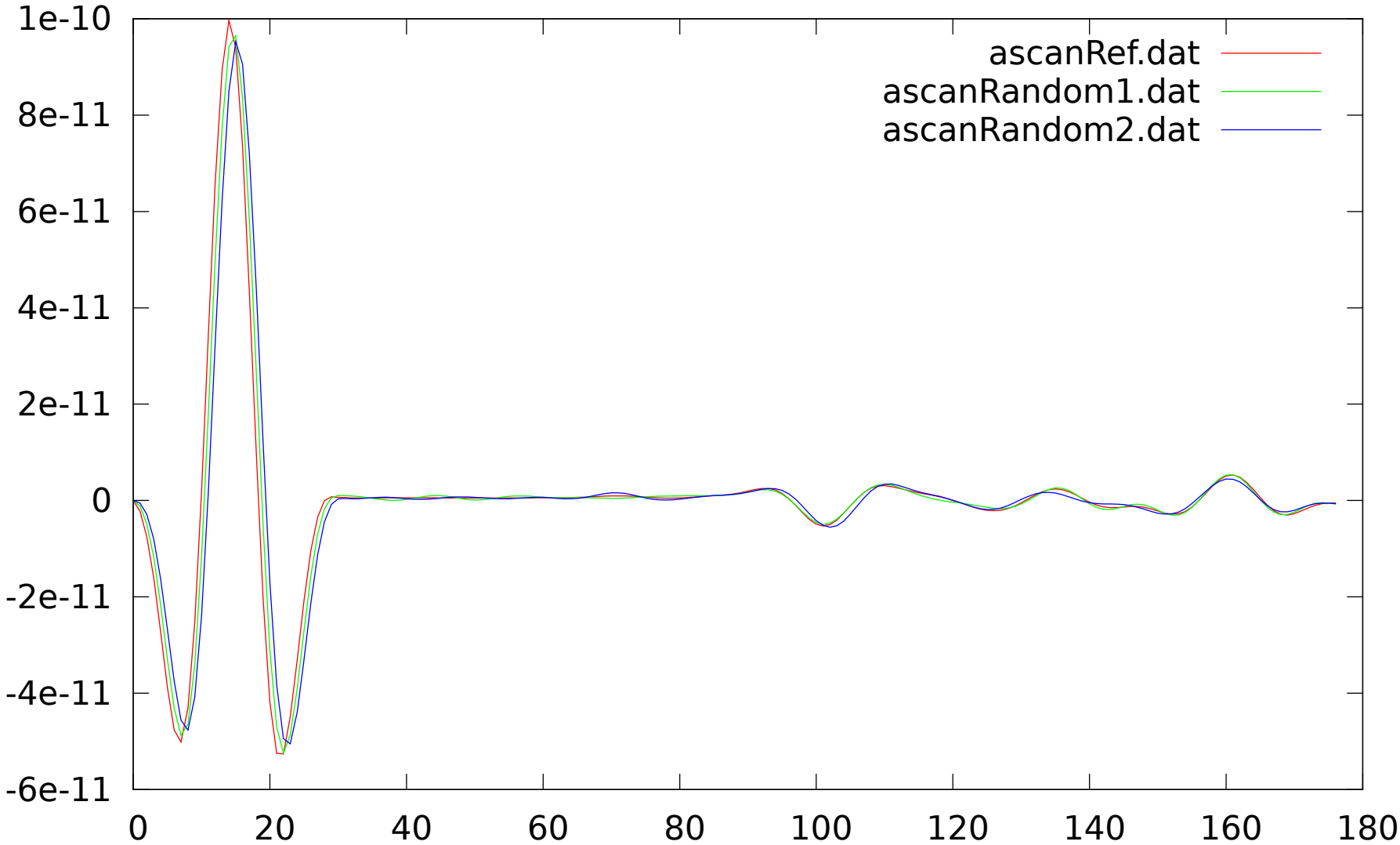
Codes Fortran 77+90

- (Athena2D) 36k lignes de code
- (Athena3D) 70k lignes

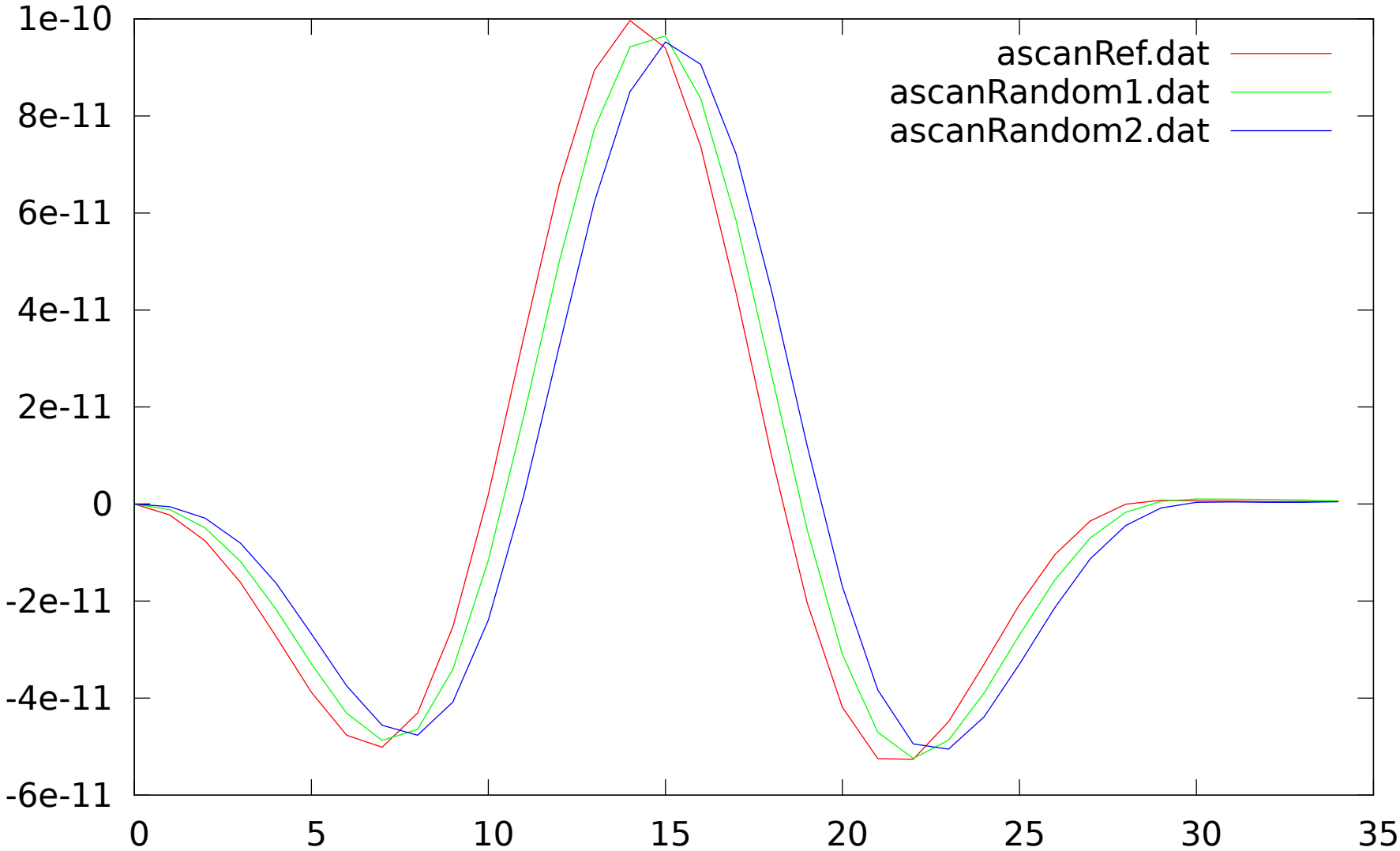
Bibliothèques externes

- (Athena2D) BLAS, LAPACK
- (Athena3D) BLAS, LAPACK, UmfPack, MPI

Analyse fine des résultats



Analyse fine des résultats



CADNA vs Verrou

avantages comparés (1 / 2)

Facilité d'utilisation

- Verrou ne nécessite pas (trop) de modifier les sources
- Verrou s'intègre facilement dans une base de non-régression
- CADNA gère plus simplement la libmath

Localisation des instabilités

- Verrou permet la localisation par essais/erreurs via les instructions début/fin d'instrumentation
- CADNA permet la localisation d'instabilités via gdb (interface graphique CADNAGrind)

CADNA vs Verrou

avantages comparés (2 / 2)

Numériquement

- Verrou permet de prendre en compte les optimisations du compilateur
- Verrou permet de mettre en évidence certains cas où la loi d'erreur n'est pas uniforme, via la comparaison Average-random
- CADNA permet l'auto-validation

Branchements synchrones / asynchrones

- Verrou est plus adapté aux branchements entre domaines de validité sans recouvrement.
- CADNA est plus adapté aux branchements entre domaines de validité avec recouvrement.

Performance

étude à mener...