Différentiation automatique et formes de Taylor en analyse statique de programmes numériques

Alexandre Chapoutot* — Matthieu Martel**

** Laboratoire MeASI - CEA Saclay*
Boîte courrier 94, F-91191 Gif-sur-Yvette cedex
alexandre.chapoutot@cea.fr

*** Laboratoire ELIAUS-DALI - Université de Perpignan Via Domitia*
52, avenue Paul Alduy, F-66860 Perpignan cedex
matthieu.martel@univ-perp.fr

RÉSUMÉ. *Des travaux récents sur l'analyse statique de programmes numériques ont montré que les techniques d'interprétation abstraite étaient adaptées à la validation de la précision des calculs en arithmétique flottante. L'utilisation des intervalles comme domaine numérique, même avec des méthodes de subdivision, induit une sur-approximation des résultats en particulier par l'existence de l'effet enveloppant (wrapping effect). Une solution utilisée pour éviter ce problème est la définition de domaines relationnels étroitement liés aux propriétés à valider. Nous allons montrer dans cet article comment des techniques de différentiation automatique peuvent être utilisées pour définir des formes de Taylor permettant de définir une nouvelle analyse statique.*

ABSTRACT. *Recent work on static analysis of numerical programs has shown that abstract interpretation is well-suited to the validation of the precision of floating-point computations. However, the non relational, interval based techniques used in this context yields some imprecise results due to the well-known wrapping effect. These over-approximations can be reduced by means of relational numerical domains which introduce some constraints between the terms of a computation. In this article, we use automatic differentiation techniques in order to define a new static analysis which relates the floating-point numbers to their error terms. This relation improves the precision of the analysis.*

MOTS-CLÉS : *interprétation abstraite, précision numérique, nombres flottants.*

KEYWORDS: *abstract interpretation, numerical precision, floating-point numbers.*

DOI:10.3166/TSI.28.503-531 © 2009 Lavoisier, Paris

RSTI - TSI – 28/2009. Supervision et sécurité en réseaux, pages 503 à 531

1. Introduction

Les règles de conception des logiciels embarqués critiques imposent, en particulier dans le domaine de l'avionique, d'apporter des garanties sur le « bon comportement » des programmes. Or, l'arithmétique à virgule flottante est de plus en plus utilisée dans ces applications, notamment à cause de la présence d'unités de calcul dédiées, ce qui nécessite la mise au point de méthodes de validation appropriées. Les techniques de test restent difficilement applicables à la détection de problèmes de précision numérique (les pires erreurs peuvent survenir pour des jeux de données très particuliers qui ne correspondent pas, par exemple, à des cas limites tels que les bornes du domaine d'une variable) et, plusieurs travaux de recherche récents ont porté sur la validation par analyse statique de la précision de ces calculs (Goubault *et al.*, 2006b; Martel, 2006; Goubault *et al.*, 2006a). Dans cet article, nous présentons une nouvelle technique pour l'étude de la précision numérique, adaptée à l'analyse statique par interprétation abstraite.

Plusieurs méthodes ont été proposées pour étudier la précision numérique des calculs réalisés par un programme (Bajard *et al.*, 1997; Martel, 2005). Cependant, les objectifs ne sont pas, en général, la validation de comportements numériques mais la représentation des valeurs réelles en machine (Potts *et al.*, 1997; Ait-Ameur, 1999), c'est-à-dire l'amélioration de la précision des résultats. L'arithmétique d'intervalles permet d'encadrer une valeur réelle par deux nombres machines. Les calculs effectués avec cette arithmétique sont garantis puisqu'à chaque opération le pire cas est considéré, d'où l'apparition de l'*effet enveloppant* cause de sur-approximations. La méthode CESTAC (Chesneaux *et al.*, 1988; Chesneaux, 1995; Vignes, 1996) (contrôle et estimation stochastique des arrondis de calculs) consiste à exécuter plusieurs fois un programme en changeant aléatoirement la manière d'arrondir les opérations, ce qui permet d'estimer le résultat réel d'un calcul. Comme toute méthode statistique, elle ne garantit les résultats qu'à probabilité près. Un autre exemple d'étude de la précision numérique qui a pour objectif la représentation des réels est la méthode CENA (Langlois, 1999) (correction des erreurs numériques d'arrondi). Elle utilise des informations données par les dérivées des fonctions calculées par un programme pour compenser les erreurs d'arrondi et approcher au mieux les calculs dans les réels. En général, ces méthodes ne sont pas bien adaptées à la validation des comportements numériques des programmes.

La théorie de l'interprétation abstraite (Cousot *et al.*, 1977; Cousot *et al.*, 1992; Cousot, 2000) permet, dans certains cas, d'apporter formellement ces garanties. Elle a été utilisée avec succès pour prouver l'absence d'erreurs à l'exécution de programmes embarqués critiques de très grande taille (Blanchet *et al.*, 2003) et, elle s'est avérée être adaptée à l'analyse de la précision numérique comme l'ont montré les travaux (Goubault, 2001; Goubault *et al.*, 2006a; Goubault *et al.*, 2006b) qui utilisent une arithmétique flottante avec erreurs permettant de calculer la distance (c'est-à-dire l'erreur) séparant le résultat flottant du résultat réel, et de tracer l'origine des erreurs. Ces informations supplémentaires sont très utiles pour la phase de correction.

Les travaux présentés dans cet article ont pour objectif d'améliorer le calcul des erreurs d'arrondi dans l'arithmétique flottante avec erreurs. Pour cela, nous allons utiliser la méthode de la différentiation automatique (Griewank, 2000; Bischof *et al.*, 2002) qui permet de calculer les dérivées d'une fonction et ainsi de mettre au point des approximations polynomiales de la fonction. Le couplage de cette méthode avec l'arithmétique d'intervalles nous permet, d'une part, de garantir les résultats des approximations et, d'autre part, de limiter l'influence de l'effet enveloppant dans le calcul des erreurs et ainsi de limiter les sur-approximations dans l'analyse de la précision numérique par interprétation abstraite. La combinaison de l'arithmétique flottante avec erreurs et de la méthode de différentiation automatique est à la base de la nouvelle arithmétique présentée dans cet article : *l'arithmétique de l'erreur différentiée*.

Le reste de cet article est organisé comme suit : La Norme IEEE754, l'analyse statique par interprétation abstraite, l'arithmétique d'intervalles, l'arithmétique flottante avec erreurs et la méthode de la différentiation automatique sont introduites à la section 2. La nouvelle arithmétique est définie à la section 3 ; elle a été implantée dans un prototype et des résultats expérimentaux sont présentés à la section 4.

2. Notions préliminaires

2.1. *Norme IEEE 754*

Dans cet article, nous supposons que l'arithmétique utilisée en machine est conforme à la norme IEEE754 (IEEE Task P754, 1985; Goldberg, 1991; Monniaux, 2007) dont les principaux aspects sont présentés ci-après. Cette norme définit l'arithmétique à virgule flottante en base 2 qui est implantée dans la majorité des processeurs actuels. Elle caractérise complètement la représentation mémoire des éléments de cette arithmétique, appelés nombres à virgule flottante, nombres flottants ou flottants. Un flottant est composé en mémoire (au niveau du bit) de trois parties : un signe s valant 1 ou -1, un exposant e compris entre e_{min} et e_{max} et une mantisse m. Le réel r associé à un nombre flottant est donné par la relation $r = s.m.2^e$.

La norme propose plusieurs types de flottants dont les plus importants sont le type simple précision et le type double précision. La précision p d'un nombre flottant correspond au nombre de bits qui compose sa mantisse. De plus, pour une précision donnée, les valeurs de e_{min} et e_{max} sont fixées. Par exemple, en double précision la mantisse est codée sur 53 bits, $e_{max} = 1023$ et $e_{min} = -e_{max} - 1$ ce qui correspond à un codage sur 10 bits de l'exposant.

Les opérations $+, -, \times, \div$ et $\sqrt{}$ ne sont pas définies en fonction des caractéristiques techniques de la machine mais suivant des contraintes mathématiques. La propriété partagée par ces opérations est celle de l'arrondi exact, c'est-à-dire que le résultat r d'une opération flottante o est équivalent au résultat r' obtenu par le calcul de o en précision infinie puis arrondi. La norme définit plusieurs modes d'arrondi dont les principaux sont : vers $+\infty$ qui donne un flottant plus grand que le réel, vers $-\infty$

qui donne un flottant plus petit que le réel et au plus près qui prend le flottant le plus proche du réel.

Une information importante sur les flottants est l'*ulp* (Unit in the Last Place) dont la valeur donne l'ordre de grandeur du dernier bit de la mantisse. L'ulp permet d'estimer la distances séparant deux nombres flottants, c'est-à-dire de majorer l'erreur d'arrondi. Pour un flottant f, l'ulp est défini par :

$$ulp(f) = 2^{-p+e-1},$$

avec p la précision du flottant et e son exposant. La majoration de l'erreur dépend du mode d'arrondi : pour les arrondis vers les infinis l'erreur d'arrondi est majorée par $ulp(f)$ tandis qu'avec le mode au plus près la majoration est de $\frac{1}{2}ulp(f)$.

Nous présentons un exemple de calcul en arithmétique flottante à la figure 1 qui montre la difficulté d'interpréter des résultats obtenus en machine. Le programme calcule une approximation de la dérivée de la fonction $\mathbf{f}(x) = 2x$ en $x = 1$. La valeur théorique de la dérivée d est égale à 2. Nous réalisons un calcul approché en machine avec des flottants en simple précision suivant la formule :

$$\mathbf{f}'(x) \approx \frac{\mathbf{f}(x+h) - \mathbf{f}(x)}{h},$$

avec des valeurs de h petites.

Quand $h = 1e^{-8}$, il y a un phénomène d'absorption qui se produit au niveau de l'expression $\mathbf{f}(x+h)$, c'est-à-dire que le calcul $1+1e^{-8}$ donne comme résultat 1. Nous obtenons alors $\mathbf{f}(x+h) = \mathbf{f}(x)$ ce qui conduit au résultat 0.0. Si $h = 1e^{-7}$, le résultat de $1 + 1e^{-7}$ est très proche de 1 ce qui engendre un autre problème, l'élimination catastrophique. Les valeurs $1+1e^{-7}$ et 1 étant très proches, quand nous les soustrayons une erreur importante survient qui est amplifiée par la division, d'où le résultat. Avec $h = 1e^{-6}$, les mêmes problèmes sont présents mais moins soulignés, tandis qu'avec $h = 1e^{-5}$ et $= 1e^{-4}$ les erreurs introduites par les calculs flottants sont encore moins importantes et donc influencent très peu le résultat.

Nous introduisons deux fonctions qui nous servirons dans la suite de cet article : $\uparrow_\circ : \mathbb{R} \rightarrow \mathbb{F}$ et $\downarrow_\circ : \mathbb{R} \rightarrow \mathbb{R}$. La fonction $\uparrow_\circ$ associe à un réel r le nombre flottant f correspondant, suivant le mode d'arrondi $\circ$ choisi. Cette fonction définit la partie représentable de r dans les flottants. La fonction $\downarrow_\circ$ calcule la partie non représentable de r dans les flottants et elle est définie par :

$$\downarrow_\circ (r) = r - \uparrow_\circ (r) .$$

2.2. *Analyse statique par interprétation abstraite*

Nous présentons brièvement dans cette section l'analyse statique par interprétation abstraite (Cousot *et al.*, 1977; Cousot *et al.*, 1992), en reprenant la présentation d'A. Miné [1]. Cette théorie permet de calculer une approximation (abstraction) des

1. Antoine Miné, cours de master, Ecole Normale Supérieure.

```
float f (float x) {
  return 2 * x;
}

float derivation (float h) {
  float x = 1;
  float d = 0.0;

  d = (f(x+h) - f(x)) / h;

  return d;
}
```

Résultats

Théorique : $d = 2$

Machine : $d = 0.0$ $(h = 1e^{-8})$

$d = 2.384186$ $(h = 1e^{-7})$

$d = 1.907349$ $(h = 1e^{-6})$

$d = 2.002716$ $(h = 1e^{-5})$

$d = 2.000332$ $(h = 1e^{-4})$

Figure 1. *Problèmes liés à l'arithmétique flottante : absorption ($h = 1e^{-8}$) et élimination catastrophique ($h = 1e^{-7}$)*

comportements d'un programme P par rapport à la description la plus précise de P, c'est-à-dire sa sémantique (concrète). Nous considérons le langage défini par la grammaire décrite à la figure 2. La règle a représente la construction des expressions arithmétiques composées de constantes réelles r, de variables x appartenant à l'ensemble fini $\mathcal{V}$ et des opérations arithmétiques $+, -, \times, \div$. Nous considérons qu'un programme P est représenté par un graphe de flot de contrôle G dont les arcs sont étiquetés par les instructions i de P, nous désignons par $\mathcal{I}$ l'ensemble de ces instructions. Les instructions sont représentées par la règles $inst$ et elles représentent soit une affectation soit une comparaison entre une variable et une constante.

$$a ::= r \mid x \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1 \mid a_0 \div a_1$$

$$inst ::= x = a \mid x \leq r \mid x < r \mid x \geq r \mid x > r$$

Figure 2. *Grammaire des opérations apparaissant sur les arcs du graphe de flot de contrôle*

Les nœuds c de G représentent les points de contrôle du programme, c'est-à-dire les états du programme après évaluation d'une instruction. Il n'y a qu'un nombre fini de points de contrôle et nous notons $\mathcal{C}$ l'ensemble de ces points. Le choix des points de contrôle est effectué lors de la phase d'analyse grammaticale et il ne change pas pendant l'analyse. Un point d'entrée e est associé à G représentant le point d'entrée du programme ainsi qu'une relation $\mathcal{P}$ qui lie les points de contrôle entre eux et les instructions, $\mathcal{P} \in \mathcal{C} \times \mathcal{C} \times \mathcal{I}$.

Nous notons σ l'environnement qui à une variable associe une valeur, c'est-à-dire $\sigma : \mathcal{V} \rightarrow \mathbb{R}$, et nous appelons Σ l'ensemble des environnements. Nous notons $\wp(E)$ l'ensemble des parties de l'ensemble E. La sémantique concrète des expressions définie l'évaluation d'une expression a dans un environnement σ en un ensemble de valeurs réelles.

La sémantique $[\![.]\!]_{\mathbb{A}} : (\mathcal{V} \to \mathbb{R}) \to \wp(\mathbb{R})$ des expressions est définie par :

$$[\![r]\!]_{\mathbb{A}}(\sigma) = \{r\}$$
$$[\![x]\!]_{\mathbb{A}}(\sigma) = \{\sigma(x)\}$$
$$[\![a_1 \diamond a_2]\!]_{\mathbb{A}}(\sigma) = \{v_1 \diamond v_2 \mid v_1 \in [\![a_1]\!]_{\mathbb{A}}(\sigma), v_2 \in [\![a_2]\!]_{\mathbb{A}}(\sigma)\}, \quad \text{avec } \diamond \in \{+, -, \times, \div\} .$$

La sémantique concrète des instructions $[\![.]\!]_{\mathbb{C}} : \wp(\mathcal{V} \to \mathbb{R}) \to \wp(\mathcal{V} \to \mathbb{R})$ est une relation entre environnements et elle est définie par :

$$[\![x = a]\!]_{\mathbb{C}}(E) = \{\sigma[x \leftarrow v] \mid \sigma \in E, v \in [\![a]\!]_{\mathbb{A}}(\sigma)\}$$
$$[\![x * r]\!]_{\mathbb{C}}(E) = \{\sigma \mid \sigma \in E, v \in [\![x]\!]_{\mathbb{A}}(\sigma), v * r\}, \quad \text{avec } * \in \{<, \leq, >, \geq\} .$$

La sémantique d'un programme est alors définie comme la plus petite solution d'un système récursif d'équations sémantiques. A chaque point de contrôle est accumulé l'ensemble des environnements rencontrés lors de toutes les exécutions, c'est-à-dire qu'à chaque point de contrôle est associé un ensemble d'environnements (c'est-à-dire un invariant). Nous notons ρ_c l'ensemble des environnements au point de contrôle c tel que :

$$\rho_c = \begin{cases} (\mathcal{V} \to \mathbb{R}) & \text{si } c = e \\ \bigcup\limits_{(c,c',i) \in \mathcal{P}} [\![i]\!]_{\mathbb{C}}(\rho_{c'}) & \text{sinon} \end{cases} .$$

Par le théorème de Tarski, nous savons que cette solution existe puisque :

– $D = \langle \wp(\mathcal{V} \to \mathbb{R}), \subseteq, \cup, \cap, \emptyset, (\mathcal{V} \to \mathbb{R}) \rangle$ est un treillis complet ;

– chaque fonction $c \mapsto \bigcup\limits_{(c,c',i) \in \mathcal{P}} [\![i]\!]_{\mathbb{C}}(\rho_{c'})$ est croissante.

La résolution de ce système se fait par itération. La figure 3 donne un exemple de programme représenté sous la forme de graphe de flot de contrôle et d'un système d'équations sémantiques.

Dans la majeure partie des cas, la sémantique concrète n'est pas calculable. Les valeurs ne sont pas représentable en mémoire et le nombre d'itérations est potentiellement infini. La théorie de l'interprétation abstraite permet alors de calculer une sur-approximation de la sémantique concrète. Nous définissons un treillis abstrait de valeurs dont un élément représente un ensemble de valeurs concrètes. Un treillis abstrait est défini par :

– $D^\sharp$ un ensemble de valeurs symboliques ;

– un ordre partiel $\sqsubseteq^\sharp$ avec un plus petit élément $\bot^\sharp$ et un plus grand élément $\top^\sharp$;

– des équivalents des opérateurs d'union $\cup^\sharp$ et d'intersection $\cap^\sharp$.

En nous appuyant sur ce treillis, nous redéfinissons les sémantiques des expressions et des instructions ce qui conduit à représenter un programme par un système d'équations sémantiques abstraites. La théorie de l'interprétation abstraite permet de s'assurer de la correction de cette abstraction en se fondant sur les correspondances de Galois. Les fonctions α, γ forme une correspondance de Galois si :

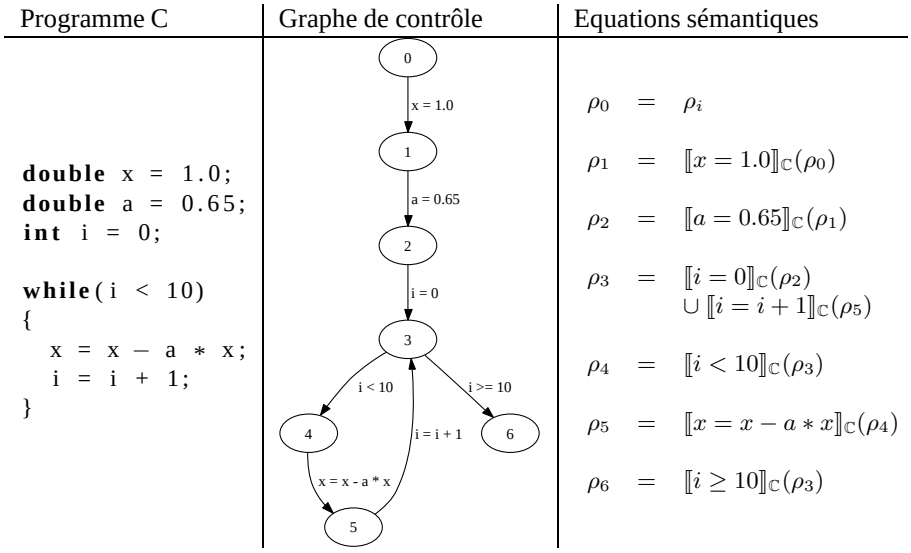


Figure 3. *Programme représenté sous forme d'un graphe de flot de contrôle et d'un système d'équations sémantiques*

– $\gamma : D^\sharp \to \wp(\mathcal{V} \to \mathbb{R})$ est croissante ;
– $\alpha : \wp(\mathcal{V} \to \mathbb{R}) \to D^\sharp$ est croissante ;
– et vérifiant $\alpha(X) \sqsubseteq^\sharp Y^\sharp \Leftrightarrow X \subseteq \gamma(Y^\sharp)$.

En se fondant sur cette correspondance, la sûreté de l'analyse est donnée par :

$$\forall c \in \mathcal{C}, \quad \rho_c \subseteq \gamma(\rho_c^\sharp) \ .$$

L'objectif de cet article est la présentation d'un nouveau treillis de valeurs définit spécialement pour l'étude de la précision numérique. Ce treillis améliore un domaine existant : le treillis des flottants avec erreurs est présenté à la section 2.4.

2.3. *L'arithmétique d'intervalles*

Une des principales difficultés en analyse numérique est la représentation des nombres réels. Une des solutions proposées pour ce problème est l'arithmétique d'intervalles (Moore, 1979; Bajard *et al.*, 1997; Rump, 1999; Revol, 2001; Jaulin *et al.*, 2001). Cette arithmétique permet de représenter en machine des nombres réels en les encadrant par des nombres machines. Par exemple, le réel $\frac{1}{3}$ peut être approché par l'intervalle $[0.33333, 0.33334]$. Nous notons $\mathbb{I}$ l'ensemble des intervalles.

Un intervalle $[a, b]$ est composé d'une borne inférieure a et d'une borne supérieure b. Une valeur réelle r est représentée par un intervalle dont la borne inférieure est $\uparrow_{-\infty}(r)$ et la borne supérieure est $\uparrow_{+\infty}(r)$. La définition des opérateurs mathématiques $+, -, \times, \div$ est étendue pour manipuler des intervalles. La figure 4 donne la sémantique $[\![.]\!]_{\mathbb{I}}$ de ces opérateurs (Moore, 1979; Revol, 2001). La somme de deux intervalles est la somme des bornes inférieures et la somme des bornes supérieures. La soustraction est la différence des bornes opposées. La multiplication nécessite de calculer toutes les combinaisons possibles des bornes pour extraire la plus petite et la plus grande valeur comme bornes de l'intervalle résultat. La division impose que le dénominateur ne contienne pas zéro pour être effectuée.

$$[\![x_1]\!]_{\mathbb{I}} = [a_1,\ b_1] \text{ et } [\![x_2]\!]_{\mathbb{I}} = [a_2,\ b_2]$$

$$[\![x_1 + x_2]\!]_{\mathbb{I}} = [a_1 + a_2,\ b_1 + b_2]$$

$$[\![x_1 - x_2]\!]_{\mathbb{I}} = [a_1 - b_2,\ b_1 - a_2]$$

$$[\![x_1 \times x_2]\!]_{\mathbb{I}} = [\min(a_1a_2,\ a_1b_2,\ b_1a_2,\ b_1b_2),\ \max(a_1a_2,\ a_1b_2,\ b_1a_2,\ b_1b_2)]$$

$$[\![\frac{1}{x_1}]\!]_{\mathbb{I}} = [\frac{1}{b_1},\ \frac{1}{a_1}] \text{ avec } 0 \notin [a_1,\ b_1]$$

Figure 4. *Définition de l'arithmétique d'intervalles*

L'ensemble des valeurs intervalles devient un ensemble partiellement ordonné, en ajoutant la relation d'ordre $\sqsubseteq_{\mathbb{I}}$ définie par :

$$[a_1, b_1] \sqsubseteq_{\mathbb{I}} [a_2, b_2] \Leftrightarrow a_1 \geq a_2 \wedge b_1 \leq b_2$$

et les opérations d'union $\sqcup_{\mathbb{I}}$ et d'intersection $\sqcap_{\mathbb{I}}$ définies par :

$$[a_1, b_1] \sqcup_{\mathbb{I}} [a_2, b_2] = [\min(a_1, a_2), \max(b_1, b_2)],$$

$$[a_1, b_1] \sqcap_{\mathbb{I}} [a_2, b_2] = [\max(a_1, a_2), \min(b_1, b_2)]\ .$$

(Cousot *et al.*, 1977) démontrent que $\langle \mathbb{I}, \sqsubseteq_{\mathbb{I}}, \sqcup_{\mathbb{I}}, \sqcap_{\mathbb{I}}, \emptyset_I, [-\infty, +\infty] \rangle$ forme un treillis complet et, démontre l'existence de la correspondance de Galois suivante :

$$(\wp(\mathbb{R}), \subseteq) \underset{\alpha_{\mathbb{I}}}{\overset{\gamma_{\mathbb{I}}}{\leftrightarrows}} (\mathbb{I}, \sqsubseteq_{\mathbb{I}})$$

avec

$$\alpha_{\mathbb{I}}(R) = [\min(R), \max(R)] \text{ et } \gamma_{\mathbb{I}}([a, b]) = \{x \mid a \leq x \leq b\}\ .$$

La sûreté de l'analyse statique par interprétation abstraite utilisant le treillis des intervalles a été démontrée dans (Cousot *et al.*, 1977).

Nous introduisons dans ce paragraphe les notations et fonctions liées à l'arithmétique d'intervalles que nous utiliserons dans la suite de cet article. Les valeurs entre crochets [ ] représenteront des valeurs intervalles par exemple $[x]$ représente un intervalle sans expliciter ses bornes et, $[a, b]$ est un intervalle dont a est la borne inférieure

et b la borne supérieure. Dans la section 3, nous utilisons le centre d'un intervalle représenté par la fonction **m** définie par :

$$\mathbf{m}([a,b]) = \frac{a+b}{2} \ . \qquad [1]$$

2.4. *Arithmétique flottante avec erreurs*

Dans cette section, nous présentons la sémantique de l'erreur globale, notée $[\![.]\!]_{\mathbb{E}}$, qui est une des deux composantes de la nouvelle arithmétique introduite dans cette article. Nous présentons tout d'abord la sémantique concrète qui permet de calculer exactement la distance séparant un résultat flottant du résultat réel, puis la sémantique abstraite qui est fondée sur l'arithmétique d'intervalles et qui permet, en pratique, de valider la précision numérique d'un programme.

Dans la suite de cet article, toutes les sémantiques que nous allons définir s'appuient sur le langage des expressions arithmétiques donné à la figure 5. Ce langage est composé de valeurs réelles r, de variables x, et des opérations $+, -, \times, \div$. La différence par rapport au langage définit à la figure 2 est que chaque élément d'une expression arithmétique est muni d'une étiquette unique ℓ permettant de l'identifier. Ces étiquettes représentent de nouveaux points de contrôle, c'est-à-dire les points importants du programme pour l'analyse de la précision numérique. Ces points de contrôle explicitent l'origine des erreurs d'arrondi et permettent ainsi de les tracer.

$$a^{\ell} ::= r^{\ell} \mid x^{\ell} \mid a_0^{\ell_0} +^{\ell} a_1^{\ell_1} \mid a_0^{\ell_0} -^{\ell} a_1^{\ell_1} \mid a_0^{\ell_0} \times^{\ell} a_1^{\ell_1} \mid a_0^{\ell_0} \div^{\ell} a_1^{\ell_1}$$

Figure 5. *Grammaire des expressions arithmétiques étiquetées*

Nous présentons la sémantique de l'erreur globale qui permet de calculer globalement l'erreur due aux arrondis pour chaque variable du programme. Il faut remarquer que nous manipulons deux types de points de contrôle : les premiers, introduits à la section 2.2, découplent un programme P en blocs élémentaires ayant une influence sur les états de P. Les seconds découpent les valeurs manipulées par P afin d'étudier l'origine des erreurs d'arrondi et ainsi d'étudier la précision numérique des calculs. Ce sont ces derniers points de contrôle que nous considérerons dans la suite de cet article. Afin de simplifier les notations nous ne considérons pas d'opérations ensemblistes comme celles introduites à la section 2.2 mais des calculs sur des valeurs. L'extension aux ensembles est naturelle et ne présente pas de difficultés particulières.

2.4.1. *Sémantique concrète*

L'objectif de l'arithmétique flottante avec erreurs est de mesurer la qualité d'un calcul flottant par rapport au même calcul réalisé avec l'arithmétique réelle. Une valeur réelle r est décomposée en deux valeurs (f, e), avec f la valeur flottante et e l'erreur d'arrondi tel que :

$$r = f + e,$$

e représente la distance séparant le flottant f du réel r. Ainsi, plus la valeur de e est petite et plus la qualité du calcul est grande.

Cette arithmétique est définie par induction suivant la structure d'une expression arithmétique a par :

$$[\![a]\!]_{\mathbb{E}} = (\mathcal{F}(a), \mathcal{E}(a)),$$

la fonction $\mathcal{F}$, donnée à la figure 6(a), correspond à l'arithmétique flottante telle qu'elle est définie dans la norme IEEE754. La fonction $\mathcal{E}$, donnée à la figure 6(b), permet de calculer l'erreur globale générée par l'expression arithmétique considérée.

La fonction $\mathcal{E}$ permet de propager les erreurs entre les différents éléments d'une expression arithmétique. De plus, conformément à la norme IEEE754, chaque opération introduit une nouvelle erreur issue de l'arrondi du résultat de cette opération. Par exemple, comme nous pouvons le voir à la figure 6(b), l'erreur pour une addition est $\mathcal{E}(a+b) = e_a + e_b + \downarrow_\circ (f_a + f_b)$: les erreurs sur les deux opérandes sont additionnées et l'erreur $\downarrow_\circ (f_a + f_b)$ due à l'addition flottante $\uparrow_\circ (f_a + f_b)$ est elle-même ajoutée au résultat. La sémantique de la soustraction est similaire à celle de l'addition. Quant à celle de la multiplication, elle découle du développement de $(f_a + e_a) \times (f_b + e_b)$. La définition du calcul de l'erreur pour l'opération de l'inverse est plus compliquée puisqu'elle fait intervenir une décomposition en série entière (Martel, 2006; Martel, 2005).

$$a = (f_a, e_a) \;\; et \;\; b = (f_b, e_b)$$

$\mathcal{F}(a+b) = \uparrow_\circ (f_a + f_b)$	$\mathcal{E}(a+b) = e_a + e_b + \downarrow_\circ (f_a + f_b)$
$\mathcal{F}(a-b) = \uparrow_\circ (f_a - f_b)$	$\mathcal{E}(a-b) = e_a - e_b + \downarrow_\circ (f_a - f_b)$
$\mathcal{F}(a \times b) = \uparrow_\circ (f_a \times f_b)$	$\mathcal{E}(a \times b) = e_a f_b + e_b f_a + e_a e_b + \downarrow_\circ (f_a \times f_b)$
$\mathcal{F}\left(\frac{1}{a}\right) = \uparrow_\circ \left(\frac{1}{f_a}\right)$	$\mathcal{E}\left(\frac{1}{a}\right) = \sum_{i=1}^{+\infty} (-1)^i \frac{e_a^i}{f_a^{i+1}} + \downarrow_\circ \left(\frac{1}{f_a}\right)$
(a) *Fonction* $\mathcal{F}$	(b) *Fonction* $\mathcal{E}$

Figure 6. *Définition des fonctions $\mathcal{F}$ et $\mathcal{E}$*

Cette sémantique permet de calculer le couple flottant et erreur à chaque point de contrôle du programme. Une instabilité numérique est détectée quand la valeur de l'erreur est importante.

2.4.2. *Sémantique abstraite*

L'objectif de la sémantique abstraite, notée $[\![.]\!]^\sharp_{\mathbb{E}}$, est d'étudier la précision numérique d'un programme pour des classes d'exécution. Nous voulons valider le comportement numérique du programme pour des plages d'entrées possibles. Pour cela, nous abstrayons les ensembles d'entrées réelles R possibles par un couple d'intervalles $([f], [e])$. L'intervalle $[f]$ est une sur-approximation de l'ensemble des flottants

représentant R en machine et l'intervalle $[e]$ est une sur-approximation de l'ensemble des erreurs $\downarrow_\circ (x),\ \forall x \in R$.

Nous étendons la fonction $\downarrow_\circ$ à des valeurs d'intervalles grâce à la fonction $\downarrow_\circ^I$ définie par l'équation [2]. Cette fonction est paramétrée par le mode d'arrondi qui détermine l'intervalle d'erreurs. Pour un intervalle $[a, b]$ avec a la borne inférieure et b la borne supérieure, l'erreur d'arrondi maximale η est donnée par la valeur de l'ulp de la plus grande borne, telle que définie à la section 2.1. L'ensemble des erreurs d'arrondi est alors donné par l'intervalle $[-\eta, \eta]$.

$$\downarrow_\circ^I ([a, b]) = ulp(\max(|a|, |b|)) \times [-1, 1]\ . \qquad [2]$$

La sémantique $[\![.]\!]_{\mathbb{E}}^\sharp$ est définie par induction sur la structure d'une expression arithmétique a par :

$$[\![a]\!]_{\mathbb{E}}^\sharp = (\mathcal{F}^\sharp(a), \mathcal{E}^\sharp(a)),$$

$\mathcal{F}^\sharp$ et $\mathcal{E}^\sharp$ sont les extensions des fonctions $\mathcal{F}$ et $\mathcal{E}$ à l'arithmétique d'intervalles.

Exemple 2.1 *Soit l'expression arithmétique, tirée de (Goubault* et al.*, 2006a),* $p = x - ax$ *avec* $x = 1$ *dans les flottants et avec une erreur comprise dans l'intervalle* $[-0.5, 0.5]$*, c'est-à-dire que* x *varie dans les réels entre* $[0.5, 1.5]$*. Nous fixons* $a = 0.65$ *et nous considérons qu'aucune erreur n'est attachée à* a*. Nous considérons, pour simplifier, que les calculs flottants n'engendrent pas de nouvelle erreur.*

$$\begin{aligned}
\mathcal{F}^\sharp(p) &= \mathcal{F}^\sharp(x) - \mathcal{F}^\sharp(ax) \\
&= [1,1] - [0.65, 0.65] \times [1,1] \\
&= [1,1] - [0.65, 0.65] \\
&= [0.35, 0.35] \\
\mathcal{E}^\sharp(p) &= \mathcal{E}^\sharp(x) - \mathcal{E}^\sharp(ax) \\
&= [-0.5, 0.5] - ([0,0] \times [1,1] + [-0.5, 0.5] \times [0.35, 0.35] + [0,0] \times [-0.5, 0.5]) \\
&= [-0.5, 0.5] - [-0.175, 0.175] \\
&= [-0.675, 0.675]
\end{aligned}$$

Le résultat réel est alors dans l'intervalle $[-0.325, 1.025]$ *alors que le flottant est* 0.35*, il y a donc une erreur importante.*

L'exemple précédent montre que les dépendances entre les valeurs affectent les résultats de l'analyse. La méthode de différentiation automatique permet de combler ce manque.

2.5. *Différentiation automatique*

Cette section présente la différentiation automatique (Griewank, 2000; Bischof *et al.*, 2002) qui, après l'arithmétique de l'erreur globale, est la deuxième composante de la nouvelle arithmétique définie à la section 3.

L'idée de la différentiation automatique est de considérer un programme comme une fonction mathématique définie par composition de fonctions élémentaires. En nous restreignant aux expressions arithmétiques, les fonctions élémentaires sont les opérations $+, -, \times, \div$ et les fonctions étudiées sont toutes les compositions possibles de ces opérations. Cette méthode s'appuie sur la règle de dérivation en chaîne : si **f** et **g** sont deux fonctions différentiables alors **f** $\circ$ **g** est différentiable telle que :

$$(\mathbf{f} \circ \mathbf{g})' = \mathbf{f}'(\mathbf{f} \circ \mathbf{g}') \ .$$

Cette règle donne la façon de calculer la dérivée de fonctions composées, par application des règles mathématiques usuelles de dérivation. La différentiation est réalisée suivant les variables du programme et nous distinguons deux types de variables : les *variables indépendantes* et les *variables dépendantes*. Les variables indépendantes sont la plupart du temps les variables d'entrée du programme et elles sont utilisées pour la différentiation. Les variables dépendantes sont celles pour lesquelles nous voulons calculer les dérivées et sont en général les sorties du programme. De plus, les variables sont dites *actives* si elles sont accompagnées d'une information de dérivée.

Nous définissons la sémantique $[\![.]\!]_{\mathbb{D}}$ associée à cette méthode suivant la structure d'une expression arithmétique a telle que :

$$[\![a]\!]_{\mathbb{D}} = (\mathcal{F}(a), \mathcal{D}(a)),$$

$\mathcal{F}$ est la fonction définie à la figure 6(a) implantant la norme IEEE 754 et $\mathcal{D}$, définie à la figure 7, correspond au calcul des dérivées en un point. Une valeur réelle est décomposée en un couple (f, d) avec f la valeur flottante et d la valeur de la dérivée au point f. Comme nous l'avons mentionné précédemment, la fonction $\mathcal{D}$ est définie par les règles de dérivation mathématique.

Cette méthode permet, par le biais des valeurs de la différentielle, une analyse de dépendance entre les variables présentes dans le programme. En effet, la différentielle est composée de dérivées partielles calculées par rapport aux variables indépendantes

$$a = (f_a, d_a) \ \ et \ \ b = (f_b, d_b)$$

$$\mathcal{D}(a + b) = d_a + d_b$$
$$\mathcal{D}(a - b) = d_a - d_b$$
$$\mathcal{D}(a \times b) = d_a f_b + d_b f_a$$
$$\mathcal{D}\left(\frac{1}{a}\right) = -\frac{d_a}{f_a^2} \ \text{ si } f_a \neq 0$$

Figure 7. *Définition de la fonction $\mathcal{D}$*

du programme. Si une dérivée partielle p associée à la variable x est nulle pour une variable active v alors nous pouvons conclure que v ne dépend pas de x. A l'inverse, si p a la plus grande valeur non nulle parmi les dérivées partielles composant la différentielle de v alors la variable x est la plus influente dans le calcul de v.

Exemple 2.2 *Reprenons l'exemple de la section 2.4 avec $p = x - ax$ et calculons la dérivée de p par rapport à x. $a = (0.65, 0)$ avec 0.65 la valeur flottante et 0 la valeur de la dérivée de a par rapport à x. $x = (1, 1)$ où la première composante est la valeur flottante et la seconde représente la valeur de la dérivée de x par rapport à x.*

$$\begin{aligned} \mathcal{D}(p) &= \mathcal{D}(x) - \mathcal{D}(ax) \\ &= 1 - (0.65 \times 1 + 0 \times 1) \\ &= 0.35 \end{aligned}$$

x a une influence dans le calcul de p et nous en déduisons qu'une erreur initiale e sur x induit une approximation du résultat de a dans les flottants de $0.35e$.

Il existe deux implantations possibles de la méthode de la différentiation automatique (Bischof *et al.*, 2002; Griewank, 2000) : par surcharge d'opérateurs et par transformation de code source. La méthode par surcharge s'appuie sur le concept objet de certains langages de programmation et elle est appliquée par exemple au langage C++ à travers la librairie ADOL-C [2]. C'est la plus simple des deux méthodes puisqu'elle nécessite uniquement la définition d'un nouveau type de données (c'est-à-dire une classe en programmation objets). L'inconvénient de cette méthode est une dégradation des performances tandis que sur son avantage est le peu de changements à effectuer dans le programme original.

La méthode de transformation de code source est utilisée dans les programmes ADIC [3] (Bischof *et al.*, 1997), ADIFOR [4] (Bischof *et al.*, 1994) ou TAPENADE [5] (Hascoët *et al.*, 2005) pour les langages C/C++ et Fortran respectivement. L'objectif est de modifier le programme original pour ajouter les instructions permettant le calcul des dérivées. Elle fait appel à des concepts issus de la compilation, par exemple la génération d'arbres de syntaxe abstraite. L'ajout de nouvelles instructions modifie la sémantique du programme et nécessite donc une grande expertise pour sa mise en œuvre. L'utilisation des techniques de compilation permet de mettre en place des analyses statiques et des optimisations qui permettent d'obtenir de meilleures performances pour le code généré.

Dans la nouvelle arithmétique définie à la section 3, nous nous situons entre les deux méthodes d'implantation. D'une part, l'analyse statique de programmes utilise des outils issus de la compilation. Nous avons alors accès à l'ensemble des informations du programme, comme par exemple la liste des variables. D'autre part, nous

2. http ://www.math.tu-dresden.de/ adol-c/
3. http ://www-fp.mcs.anl.gov/adic/
4. http ://www-unix.mcs.anl.gov/autodiff/ADIFOR/
5. http ://www-sop.inria.fr/tropics/tapenade.html

définissons la sémantique des expressions arithmétiques pour analyser la précision numérique et nous ne modifions pas le programme pour ajouter les expressions de calculs des dérivées. Nous nous situons ainsi dans la cas de l'implantation par surcharge des opérateurs.

3. Arithmétique de l'erreur différentiée

Dans cette section, nous présentons une nouvelle sémantique, notée $[\![.]\!]_{\mathbb{ED}}$, fondée sur les méthodes de différentiation automatique et de l'erreur globale introduites aux sections 2.4 et 2.5. Nous introduisons tout d'abord à la section 3.1 une première sémantique, notée $[\![.]\!]^1_{\mathbb{ED}}$ permettant de calculer des approximations linéaires de l'erreur, puis nous montrerons à la section 3.2 comment cette sémantique peut être étendue pour réaliser des approximations polynomiales fondée sur des polynômes de Taylor. Nous détaillerons une extension fondée sur des polynômes de Taylor du second ordre, notée $[\![.]\!]^2_{\mathbb{ED}}$.

Conformément à la théorie de l'interprétation abstraite (Cousot *et al.*, 1977; Cousot *et al.*, 1992), nous allons tout d'abord définir des sémantiques concrètes pour $[\![.]\!]^1_{\mathbb{ED}}$ et $[\![.]\!]^2_{\mathbb{ED}}$, c'est-à-dire des sémantiques spécifiant exactement le comportement d'un programme au cours d'une exécution, puis nous donnerons des versions abstraites de ces sémantiques dans lesquelles les ensembles de valeurs sont abstraits par des intervalles afin de pouvoir raisonner, au cours d'une analyse statique, sur des ensembles d'exécutions.

3.1. *Différentiation au premier ordre*

3.1.1. *Sémantique concrète*

Nous présentons maintenant la sémantique qui permet d'approcher linéairement la fonction $\mathcal{E}$ introduite à la section 2.4 pour chaque variable du programme. Nous partons du constat qu'à chaque point de contrôle ℓ d'une expression arithmétique (cf. figure 5) une nouvelle erreur est introduite. L'erreur introduite au point de contrôle ℓ sera notée o^ℓ. Elle correspond soit à l'erreur de représentation d'une constante, soit à l'erreur d'arrondi du résultat d'une opération. $\mathcal{E}$ (cf. figure 6(b)) est une fonction des o^ℓ dont le résultat est l'erreur globale. Nous allons alors appliquer la méthode de différentiation automatique à la fonction $\mathcal{E}$ par rapports aux o^ℓ. Les o^ℓ sont associées aux variables indépendantes de la différentiation automatique.

Dans la sémantique concrète $[\![.]\!]^1_{\mathbb{ED}}$, une valeur réelle r est représentée par un triplet (f, e, d) avec f la valeur flottante, e l'erreur globale et d la différentielle de e par rapport aux variables o^ℓ. La sémantique $[\![.]\!]^1_{\mathbb{ED}}$ est définie suivant la structure d'une expression arithmétique a telle que

$$[\![a]\!]^1_{\mathbb{ED}} = (\mathcal{F}(a), \mathcal{E}(a), \mathcal{D}^1_{\mathcal{E}}(a)),$$

$\mathcal{F}$ l'implantation de la norme IEEE 754 (cf. figure 6(a)) et $\mathcal{E}$ la fonction de calcul de l'erreur globale (cf. figure 6(b)). La fonction $\mathcal{D}^1_{\mathcal{E}}$ est la composition de la fonction $\mathcal{E}$ et de la fonction $\mathcal{D}$ dont la définition est donnée à la figure 8, elle calcule le gradient de $\mathcal{E}$. Nous obtenons ainsi un gradient dont les composantes sont de la forme $\partial\mathcal{E}/\partial o^\ell$.

La sémantique d'une constante réelle c au point de contrôle ℓ est donnée par le triplet $\left(\uparrow_\circ (c), \downarrow_\circ (c), \frac{\downarrow_\circ (c)}{\partial o^\ell}\right)$. L'élément $\uparrow_\circ (c)$ est la partie représentable de c dans les flottants, l'élément $\downarrow_\circ (c)$ est l'erreur de représentation et l'élément $\frac{\downarrow_\circ (c)}{\partial o^\ell}$ est le vecteur dont toutes les composantes sont nulles sauf la ℓ-ième qui vaut 1. Comme pour la fonction $\mathcal{D}$, la définition de la fonction $\mathcal{D}^1_{\mathcal{E}}$ s'appuie sur les règles de dérivation mathématique. Pour les opérations d'addition et de soustraction nous obtenons des combinaisons linéaires des vecteurs de dérivées partielles des opérandes auxquelles nous ajoutons le vecteur associé à la nouvelle erreur d'arrondi. Le calcul pour l'opération de multiplication se fonde sur la formule $ax + by + xy$ avec a et b des constantes et, x et y les variables suivant lesquelles nous différentions. Nous obtenons la différentielle $ax' + by' + x'y + xy' = x'(a + y) + y'(b + x)$ et, en posant $a = f_a$, $b = f_b$, $x = e_a$, $y = e_b$, $x' = d_a$ et $y' = d_b$, nous obtenons la définition de la figure 8. Pour l'opération d'inversion, nous différentions terme à terme la fonction $\mathcal{E}$ associée à l'opération d'inversion définie à la figure 6(b).

$$a = (f_a, e_a, d_a) \;\; et \;\; b = (f_b, e_b, d_b)$$

$$\mathcal{D}^1_{\mathcal{E}}(a+b) = d_a + d_b + \frac{\partial \downarrow_\circ (f_a + f_b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^1_{\mathcal{E}}(a-b) = d_a - d_b + \frac{\partial \downarrow_\circ (f_a - f_b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^1_{\mathcal{E}}(a\times b) = d_a(f_b + e_b) + d_b(f_a + e_b) + \frac{\partial \downarrow_\circ (f_a \times f_b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^1_{\mathcal{E}}\left(\frac{1}{a}\right) = \sum_{i=1}^{+\infty}(-1)^i \frac{1}{f^{i+1}} e^{i-1} d_a + \frac{\partial \downarrow_\circ (\frac{1}{f_a})}{\partial o^{\ell_i}}$$

Figure 8. *Définition de la fonction* $\mathcal{D}^1_{\mathcal{E}}$

Dans cette sémantique, le calcul des dérivées partielles ne permet qu'une analyse de dépendance puisque le terme d'erreur est calculé dans les réels. Par contre, nous nous servirons de leurs valeurs dans la sémantique abstraite pour calculer des ensembles d'erreurs, représentés par des intervalles, mais en réduisant de manière significative l'effet enveloppant.

3.1.2. *Sémantique abstraite*

La sémantique abstraite, notée $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$, permet, comme la sémantique abstraite $[\![.]\!]^{\sharp}_{\mathbb{E}}$ (cf. section 2.4), d'évaluer la précision numérique pour des classes d'exécutions. Nous

présentons dans cette section la manière dont nous calculons des approximations linéaires garanties de l'erreur avant de définir formellement $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$.

L'analyse par intervalles (Moore, 1979; Jaulin *et al.*, 2001) introduit la notion de *fonctions d'inclusion*. A une fonction mathématique μ est associée une fonction informatique ι calculée à l'aide de l'arithmétique d'intervalles. Cette fonction ι est une fonction d'inclusion si l'image d'un intervalle $[x]$ par μ est contenue dans l'image de $[x]$ par ι. La fonction d'inclusion naturelle est celle définie directement à partir des opérations de l'arithmétique d'intervalles. Nous notons $[\mathbf{f}]_n$ la fonction d'inclusion naturelle associée à la fonction mathématique **f**.

Nous illustrons la fonction d'inclusion naturelle par un exemple tiré de (Jaulin *et al.*, 2001). Nous calculons les fonctions d'inclusion sur l'intervalle $[-1, 1]$ de $\mathbf{f}_1$ et $\mathbf{f}_2$ deux fonctions équivalentes, définies par :

$$\begin{aligned} \mathbf{f}_1(x) &= x^2 + x \ , \\ \mathbf{f}_2(x) &= \left(x + \frac{1}{2}\right)^2 - \frac{1}{4} \ . \end{aligned}$$

La figure 9 donne une représentation graphique de l'amplitude des intervalles résultat de l'évaluation de $[\mathbf{f}_1]_n$ et $[\mathbf{f}_2]_n$. Nous avons $[\mathbf{f}_1]_n([-1, 1]) = [-2, 2]$ où nous avons représenter la puissance 2 par une multiplication. Nous avons aussi $[\mathbf{f}_2]_n([-1, 1]) = [-\frac{1}{4}, 2]$, qui est de plus le résultat mathématique exact de l'image de $[-1, 1]$ par **f**.

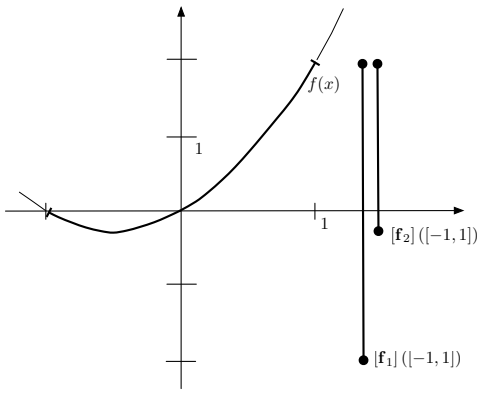


Figure 9. *Fonctions d'inclusion naturelles de $\mathbf{f}_1$ et $\mathbf{f}_2$*

Il est cependant possible de définir d'autres fonctions d'inclusion permettant de limiter l'influence, sur le résultat, de la forme d'une expression évaluée dans les intervalles. Le théorème des accroissements finis dit : "Soit **f** une fonction différentiable sur l'intervalle $[a, b]$, de différentielle **g**, alors il existe c dans $]a, b[$ tel que :

$$\mathbf{f}(b) = \mathbf{f}(a) + \mathbf{g}(c)(b - a) \ .$$

En généralisant, nous obtenons :

$$\forall x \in [a,b],\ \exists c \in]a,b[,\ \mathbf{f}(x) = \mathbf{f}(m) + \mathbf{g}(c)(x-m),$$

m est le milieu de l'intervalle $[a,b]$. Ces théorèmes nous affirment l'existence de cette valeur c mais pas le moyen de la calculer. En utilisant l'arithmétique d'intervalles pour calculer **g** nous obtenons l'ensemble des dérivées de **f** sur $[a,b]$ et nous obtenons la relation :

$$\forall x \in [a,b],\ \mathbf{f}(x) \in \mathbf{f}(m) + [\mathbf{g}]_n([a,b])(x-m)\ .$$

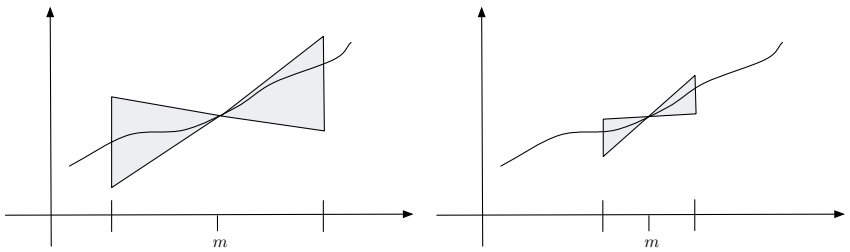
Par extension, nous obtenons :

$$\mathbf{f}([a,b]) \subseteq \mathbf{f}(m) + [\mathbf{g}]_n([a,b])([a,b]-m)\ .$$

Nous pouvons alors définir la fonction d'inclusion centrée, notée $[\mathbf{f}]_c$ pour la fonction mathématique **f**, définie pour toute fonction **f** différentiable sur un intervalle $[x]$ dont le milieu est noté m et de différentielle **g**. Nous avons :

$$[\mathbf{f}]_c([x]) = \mathbf{f}(m) + [\mathbf{g}]_n([x])([x]-m)\ . \qquad [3]$$

La figure 10 donne deux interprétations graphiques de la fonction d'inclusion centrée. Suivant la largeur de l'intervalle d'étude de la fonction, nous obtenons une approximation plus ou moins précise.



(a) *Intervalle d'étude large* (b) *Intervalle d'étude moins large*

Figure 10. *Fonction d'inclusion centrée*

Cette définition peut être étendue sans difficulté à des fonctions manipulant des valeurs vectorielles. Grâce à cette nouvelle fonction d'inclusion nous pouvons définir un nouveau calcul d'ensemble d'erreurs en prenant comme fonction **f** la fonction $\mathcal{E}$ et comme intervalle $[x]$ le vecteur d'intervalles d'erreurs élémentaires, noté $\vec{O}$.

Dans la sémantique abstraite, un ensemble de valeurs réelles R est représenté par un triplet $([f], m, [d])$ avec $[f]$ l'intervalle de flottants, m le centre de l'intervalle d'erreurs et $[d]$ le vecteur de dérivées partielles. La sémantique abstraite est définie sur la structure d'une expression arithmétique a telle que :

$$[\![a]\!]^{1\sharp}_{\mathbb{ED}} = (\mathcal{F}^\sharp(a), \mathcal{E}^\sharp_m(a), \mathcal{D}^\sharp_{\mathcal{E}}(a))\ .$$

La fonction $\mathcal{F}^\sharp$ est l'extension de la fonctions $\mathcal{F}$ à l'arithmétique d'intervalles. La fonction $\mathcal{E}_m^\sharp$, définie à la figure 11, est une adaptation de la fonction $\mathcal{E}^\sharp$ au calcul de la valeur centrée de l'erreur globale. Elle suit les mêmes règles de calcul que $\mathcal{E}^\sharp$ mais en convertissant systématiquement tous les intervalles en une valeur unique qui est leur centre en utilisant la fonction **m** définit à l'équation [1].

$$a = ([f_a], m_a, [d_a]) \;\; et \;\; b = ([f_b], m_b, [d_b])$$

$$\mathcal{E}_m^\sharp(a+b) = m_a + m_b + \mathbf{m}(\downarrow_\circ^I ([f_a] + [f_b]))$$

$$\mathcal{E}_m^\sharp(a-b) = m_a - m_b + \mathbf{m}(\downarrow_\circ^I ([f_a] - [f_b]))$$

$$\mathcal{E}_m^\sharp(a \times b) = \mathbf{m}(m_a[f_b]) + \mathbf{m}(m_b[f_a]) + m_a m_b + \mathbf{m}(\downarrow_\circ^I ([f_a] \times [f_b]))$$

$$\mathcal{E}_m^\sharp\left(\frac{1}{a}\right) = \sum_{i=1}^{+\infty} (-1)^i \frac{m_a^i}{[f_a]^{i+1}} + \mathbf{m}\left(\downarrow_\circ^I \left(\frac{1}{[f_a]}\right)\right)$$

Figure 11. *Définition de la fonction $\mathcal{E}_m^\sharp$*

La fonction $\mathcal{D}_\mathcal{E}^\sharp$ est une extension de la fonction $\mathcal{D}_\mathcal{E}$ à l'arithmétique d'intervalles où chaque occurrence d'une variable d'erreurs e est remplacée par un calcul de l'approximation linéaire définie par la relation [4]. Cette substitution permet de calculer l'ensemble des dérivées premières de l'intervalle d'erreurs et ainsi permet de garantir les approximations. $\dot{\mathbf{m}}(\vec{O})$ représente l'application de la fonction **m** sur toutes les composantes de $\vec{O}$. Nous définissons la fonction $\mathcal{A}^1$, qui calcule l'intervalle résultat de l'approximation linéaire de la fonction $\mathcal{E}^\sharp$ suivant un centre m et le vecteur de dérivées intervalles $[d]$, par :

$$\mathcal{A}^1(m, [d]) = m + [d](\vec{O} - \dot{\mathbf{m}}(\vec{O})) \; . \qquad [4]$$

L'ensemble réel R associé à la valeur abstraite $([f], m, [d])$ est égal à :

$$R = [f] + \mathcal{A}^1(m, [d]) \; .$$

Exemple 3.1 *Reprenons l'exemple de la section 2.4 avec $p = x - ax$, $x = ([1,1], 0, [1,1]_x)$ où 0 est le milieu de l'intervalle d'erreur initiale $[-0.5, 0.5]$, et $[1,1]$ est la valeur de la dérivée de e_x par rapport à l'erreur initiale. La valeur de a est $a = ([0.65, 0.65], 0, [1,1]_a)$ (erreur initiale nulle). Nous considérons pour simplifier que les calculs flottants n'engendrent pas de nouvelles erreurs. La notation x_y représente une valeur d'erreur ou de dérivée x associée à la variable y.*

$$\begin{aligned}
\mathcal{E}_m^\sharp(p) &= \mathcal{E}_m^\sharp(x) - (\boldsymbol{m}(\mathcal{E}_m^\sharp(a) \times \mathcal{F}^\sharp(x)) + \boldsymbol{m}(\mathcal{E}_m^\sharp(x) \times \mathcal{F}^\sharp(a)) + \mathcal{E}_m^\sharp(a) \times \mathcal{E}_m^\sharp(x)) \\
&= 0 - (\boldsymbol{m}(0 \times [1,1]) + \boldsymbol{m}(0 \times [0.65, 0.65]) + 0 \times 0) \\
&= 0
\end{aligned}$$

Nous savons par le calcul des dérivées que p a une erreur qui dépend de l'erreur sur a et de l'erreur sur x.

$$\begin{aligned}\mathcal{D}_{\mathcal{E}}^{1\sharp}(p) &= \mathcal{D}_{\mathcal{E}}^{1\sharp}(x) - (\mathcal{D}_{\mathcal{E}}^{1\sharp}(a) \times \mathcal{F}^{\sharp}(x) + \mathcal{D}_{\mathcal{E}}^{1\sharp}(x) \times \mathcal{F}^{\sharp}(a) + \mathcal{D}_{\mathcal{E}}^{1\sharp}(x) \times \mathcal{E}^{\sharp}(a) \\ &\quad + \mathcal{D}_{\mathcal{E}}^{1\sharp}(a) \times \mathcal{E}^{\sharp}(x)) \\ &= [1,1]_x - ([1,1]_a \times [1,1] + [1,1]_x \times [0.65, 0.65] + [1,1]_x \times 0 \\ &\quad + [1,1]_a \times [-0.5, 0.5]) \\ &= [1,1]_x - ([0.5, 1.5]_a, [0.65, 0.65]_x) \\ &= ([-1.5, -0.5]_a, [0.35, 0.35]_x)\end{aligned}$$

Nous obtenons alors un ensemble d'erreurs avec $\otimes$ *le produit scalaire et* $(0_a, [-0.5, 0.5]_x)$ *le vecteur d'erreurs initiales :*

$$\begin{aligned}\mathcal{A}^1(\mathcal{E}_m^{\sharp}(p), \mathcal{D}_{\mathcal{E}}^{1\sharp}(p)) &= \mathcal{E}_m^{\sharp}(p) + \mathcal{D}_{\mathcal{E}}^{1\sharp}(a) \otimes (0_a, [-0.5, 0.5]_x) \\ &= 0 + ([-1.5, -0.5]_a, [0.35, 0.35]_x) \otimes (0_a, [-0.5, 0.5]_x) \\ &= [-0.175, 0.175]\end{aligned}$$

Au final, nous avons un ensemble réel R *compris dans l'intervalle* $[0.175, 0.425]$ *avec le flottant toujours égal à* 0.35*.*

Comme nous l'avons vu à la section 2.2, dans la présentation générale de la théorie de l'interprétation abstraite, il est nécessaire de munir notre ensemble de valeurs d'une structure de treillis. Nous définissons maintenant le treillis des flottants avec erreurs différentiées au premier ordre. La relation d'ordre $\sqsubseteq_{\mathbb{ED}}^{1\sharp}$ est définie par :

$$([f_1], m_1, [d_1]) \sqsubseteq_{\mathbb{ED}}^{1\sharp} ([f_2], m_2, [d_2]) \Leftrightarrow \begin{cases} [f_1] \sqsubseteq_{\mathbb{I}} [f_2] \\ \wedge \\ \mathcal{A}^1(m_1, [d_1]) \sqsubseteq_{\mathbb{I}} \mathcal{A}^1(m_2, [d_2]) \end{cases} .$$

Les opérations d'union $\sqcup_{\mathbb{ED}}^{1\sharp}$ et d'intersection $\sqcap_{\mathbb{ED}}^{1\sharp}$ sont définies par :

$$\begin{aligned}&([f_1], m_1, [d_1]) \sqcup_{\mathbb{ED}}^{1\sharp} ([f_2], m_2, [d_2]) = \\ &\qquad \left([f_1] \sqcup_{\mathbb{I}} [f_2], \mathbf{m}(\mathcal{A}^1(m_1, [d_1]) \sqcup_{\mathbb{I}} \mathcal{A}^1(m_2, [d_2])), [d_1] \dot{\sqcup}_{\mathbb{I}} [d_2]\right) ,\end{aligned}$$

$$\begin{aligned}&([f_1], m_1, [d_1]) \sqcap_{\mathbb{ED}}^{1\sharp} ([f_2], m_2, [d_2]) = \\ &\qquad \left([f_1] \sqcap_{\mathbb{I}} [f_2], \mathbf{m}(\mathcal{A}^1(m_1, [d_1]) \sqcap_{\mathbb{I}} \mathcal{A}^1(m_2, [d_2])), [d_1] \dot{\sqcap}_{\mathbb{I}} [d_2]\right) .\end{aligned}$$

Les opérations $\dot{\sqcup}_{\mathbb{I}}$ et $\dot{\sqcap}_{\mathbb{I}}$ représentent l'application composante par composante des opérateurs $\sqcup_{\mathbb{I}}$ et $\sqcap_{\mathbb{I}}$. Les trois fonctions liées à la structure d'ordre du treillis des flottants avec erreurs différentiées nécessitent l'évaluation de l'intervalle d'erreurs.

3.2. *Différentiation d'ordre supérieur*

La fonction d'inclusion centrée se fonde sur le théorème des accroissements finis. La définition de nouvelles fonctions d'inclusion se fondant sur la généralisation de ce théorème aux fonctions n fois différentiables est naturelle. La formule de Taylor-Lagrange est alors la base de la définition des fonctions d'inclusion de Taylor. De plus, la différentiation d'ordre supérieur permet de prendre en compte beaucoup plus d'informations de dépendance, ce qui permet de définir des approximations plus précises. Nous montrons les possibilités d'extension de la sémantique précédente par application successive de la différentiation automatique afin de calculer des dérivées d'ordre n.

La fonction $\mathcal{D}$ peut être appliquée sur la fonction $\mathcal{D}^1_{\mathcal{E}}$ pour donner la fonction $\mathcal{D}^2_{\mathcal{E}}$ calculant les dérivées secondes de la fonction $\mathcal{E}$ par rapport aux o^ℓ. En appliquant à nouveau la fonction $\mathcal{D}$ sur la fonction $\mathcal{D}^2_{\mathcal{E}}$, nous pouvons calculer les dérivées troisièmes et ainsi de suite pour calculer les dérivées n-ième.

Grâce à la connaissance des dérivées de tous ordres, nous pouvons définir une fonction d'inclusion fondée sur des formes de Taylor (Neumaier, 2003; Jaulin *et al.*, 2001). Le théorème des accroissements finis se généralise aux fonction différentiables n fois par la formule de Taylor-Lagrange (Rudin, 1976). Et, en procédant de la même manière qu'à la section 3.1, nous pouvons définir une fonction d'inclusion de Taylor, notée $[\mathbf{f}]_T$ telle que :

$$[\mathbf{f}]_T([x]) = \mathbf{f}(m) + \ldots + \mathbf{f}^{n-1}(m)\frac{([x]-m)^{n-1}}{(n-1)!} + [\mathbf{f}^n]_n([x])\frac{([x]-m)^n}{n!}\ . \qquad [5]$$

Cette définition s'étend aussi sans difficulté à des valeurs vectorielles. L'avantage d'une telle fonction d'inclusion est de n'utiliser l'arithmétique d'intervalles qu'à l'ordre n, en général sur des petits ensembles de valeurs, et donc de limiter grandement l'effet enveloppant. Cependant, le compromis entre précision et efficacité de l'analyse doit être pris en considération, c'est la raison qui nous a conduit à n'expliciter qu'une extension à l'ordre 2 de la sémantique de la section 3.1.

Nous présentons une extension à l'ordre 2, notée $[\![.]\!]^2_{\mathbb{ED}}$. Une valeur réelle r dans la sémantique concrète est représentée par un quadruplet (f, e, j, h) où f et e représentent respectivement la partie représentable et non représentable de r et où j et h représentent les vecteurs de dérivées partielles au premier et au second ordre de l'erreur globale suivant les o^ℓ. La sémantique $[\![.]\!]^2_{\mathbb{ED}}$ est définie sur la structure d'une expression arithmétique a par :

$$[\![a]\!]^2_{\mathbb{ED}} = (\mathcal{F}(a), \mathcal{E}(a), \mathcal{D}^1_{\mathcal{E}}(a), \mathcal{D}^2_{\mathcal{E}}(a)),$$

$\mathcal{F}(a)$, $\mathcal{E}(a)$ et $\mathcal{D}^1_{\mathcal{E}}$ sont les mêmes que précédemment et sont respectivement définies aux figures 6(a), 6(b) et 8. La fonction $\mathcal{D}^2_{\mathcal{E}}$ est définie à la figure 12. Sa définition est obtenue de la même manière que la définition de la fonction $\mathcal{D}^1_{\mathcal{E}}$. La difficulté vient du calcul des dérivées secondes à partir du produit de deux vecteurs de dérivées premières. Il faut réaliser une multiplication matricielle entre les deux vecteurs afin d'obtenir toutes les combinaisons linéaires possibles.

$$a = (f_a, e_a, j_a, h_a) \;\; et \;\; b = (f_b, e_b, j_b, h_b)$$

$$\mathcal{D}^2_{\mathcal{E}}(a+b) = h_a + h_b$$

$$\mathcal{D}^2_{\mathcal{E}}(a-b) = h_a - h_b$$

$$\mathcal{D}^2_{\mathcal{E}}(a \times b) = h_a f_b + h_b f_a + j_b j_a + e_a j_b + j_a j_b + e_b j_a$$

$$\mathcal{D}^2_{\mathcal{E}}\left(\frac{1}{a}\right) = \sum_{i=1}^{+\infty} (-1)^i \frac{i}{f_a^{i+1}} \left(h_a e_a^{i-1} + (i-1) j_a^2 e^{i-2}\right)$$

Figure 12. *Définition de la fonction $\mathcal{D}^2_{\mathcal{E}}$*

Comme pour la sémantique $[\![.]\!]^1_{\mathbb{ED}}$, le calcul des dérivées premières et secondes ne permettent que de faire une analyse de dépendances. En effet, la sémantique concrète calcule exactement les erreurs d'arrondi. Les valeurs des dérivées apportent une information sur l'influence d'une variable sur le calcul de l'erreur et aucune approximation n'est nécessaire. Les dérivées d'ordre 2 permettent de connaître l'influence des combinaisons d'erreurs élémentaires, introduites par l'opération de multiplication en particulier, sur la valeur de l'erreur globale. Ceci, nous permet d'étudier les erreurs d'ordre supérieur, erreurs résultats du produit de deux erreurs, qui ont dans certains cas une influence non négligeable sur le calcul de l'erreur.

La sémantique abstraite, notée $[\![.]\!]^{2\sharp}_{\mathbb{ED}}$, utilise la fonction d'inclusion de Taylor d'ordre 2 pour calculer des ensembles d'erreurs. Cette fonction est donnée par l'équation [6] et, elle induit un changement dans le calcul des dérivées premières définies à la section 3.1, puisque ces dérivées ne doivent plus être calculées sur la totalité de l'intervalle d'erreurs mais uniquement au centre. Nous définissons à la figure 13 la fonction $\mathcal{D}^{1\sharp}_{m\mathcal{E}}$ calculant les dérivées d'ordre 1 au niveau du centre de l'intervalle d'erreur, c'est-à-dire suivant les valeurs de la fonction $\mathcal{E}^{\sharp}_m$. Cette définition est construite de la même manière que la fonction $\mathcal{E}^{\sharp}_m$ (cf. figure 11) et le résultat de chaque opération mettant en jeu un intervalle est réduit en son centre grâce à la fonction **m**. La notation $\mathbf{f}^{(i)}$ représente la differentielle d'ordre i de la fonction **f**.

$$[\mathbf{f}]_{T2}([x]) = \mathbf{f}(m) + \mathbf{f}^{(1)}(m)([x]-m) + \frac{1}{2}([x]-m)[\mathbf{f}^{(2)}]_n([x])([x]-m)\ . \qquad [6]$$

Un ensemble de valeurs réelles R est représenté dans la sémantique abstraite $[\![.]\!]^{2\sharp}_{\mathbb{ED}}$ par un quadruplet $([f], m, j, [h])$ avec $[f]$ l'ensemble des valeurs flottantes associées à R, m la valeur centrée de l'erreur, j le vecteur de dérivées partielles premières calculé au point m et h la matrice de dérivées secondes. Les dérivées secondes sont représentées sous forme de matrices où les lignes et les colonnes représentent les points de contrôle et où la valeur de la case d'indices i, j représente la dérivée seconde de la fonction $\mathcal{E}$ par rapport aux erreurs élémentaires o^{ℓ_i} et o^{ℓ_j}. Cette dérivée représente la contribution du produit de o^{ℓ_i} et de o^{ℓ_j} dans la valeur de l'erreur globale.

$$a = ([f_a], m_a, j_a, [h_a]) \;\; et \;\; b = ([f_b], m_b, j_b, [h_b])$$

$$\mathcal{D}^{1\sharp}_{m\mathcal{E}}(a+b) = j_a + j_b + \frac{\partial \downarrow_\circ (a+b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^{1\sharp}_{m\mathcal{E}}(a-b) = j_a - j_b + \frac{\partial \downarrow_\circ (a-b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^{1\sharp}_{m\mathcal{E}}(a\times b) = \dot{\mathbf{m}}(j_a[f_b]) + \dot{\mathbf{m}}(j_b[f_a]) + m_a j_b + m_b j_a + \frac{\partial \downarrow_\circ (a\times b)}{\partial o^{\ell_i}}$$

$$\mathcal{D}^{1\sharp}_{m\mathcal{E}}\left(\frac{1}{a}\right) = \sum_{i=1}^{+\infty}(-1)^i \dot{\mathbf{m}}\left(\frac{i}{[f_a]^{i+1}}\right) m^{i-1} j_a + \frac{\partial \downarrow_\circ (\frac{1}{a})}{\partial o^{\ell_i}}$$

Figure 13. *Définition de la fonction* $\mathcal{D}^{1\sharp}_{m\mathcal{E}}$

La sémantique $[\![.]\!]^{2\sharp}_{\mathbb{ED}}$ est définie sur la structure d'une expression arithmétique a telle que :

$$[\![a]\!]^{2\sharp}_{\mathbb{ED}} = (\mathcal{F}^\sharp(a), \mathcal{E}^\sharp_m(a), \mathcal{D}^{1\sharp}_{m\mathcal{E}}(a), \mathcal{D}^{2\sharp}_{\mathcal{E}}(a)) \; .$$

Comme pour la sémantique abstraite $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$, la fonction $\mathcal{D}^{2\sharp}_{\mathcal{E}}$ est une extension de la fonction $\mathcal{D}^2_{\mathcal{E}}$ à l'arithmétique d'intervalles où chaque occurrence d'une variable d'erreur est remplacée par le calcul de l'approximation polynomiale de l'erreur. Cette définition nous permet de calculer l'ensemble des dérivées partielles secondes sur l'intervalle d'erreur et elle nous permet d'avoir des approximations garanties. Le domaine est défini de manière analogue au treillis des flottants avec erreurs différentiées au premier ordre.

4. Résultats expérimentaux

Nous avons implanté la sémantique de l'erreur différentiée à l'ordre 1 et 2 dans un prototype écrit en OCaml [6] permettant d'analyser un noyau de langage impératif basé sur le langage C. Nous obtenons des résultats qui confirment l'intérêt de la composition de sémantiques dans l'étude de la précision numérique. Nous illustrons les bonnes propriétés de la sémantique $[\![.]\!]_{\mathbb{ED}}$ par deux exemples. Le premier exemple permet d'apprécier le gain de précision des résultats entre un calcul d'erreur dans les intervalles et un autre avec notre nouvelle sémantique. Cet exemple est basé sur un calcul de racine carré, par une méthode de Newton. Le second exemple a été construit, à partir de la suite $x_{n+1} = x_n - ax_n$, et met en évidence l'intérêt d'une différentiation de l'erreur au second ordre.

Dans ces deux exemples, nous avons analysé les programmes en utilisant une arithmétique flottante multiprécision pour calculer les erreurs. L'arithmétique multipréci-

6. http ://caml.inria.fr/

```
double xn = 0.1;
double xn1 = 0.0;
double a = [25, 25]_[-0.01, 0.01];
int cond = 0;
double temp = 0.0;
double res = 0.0;

while (cond < 1) {
  xn1 = 0.5 * xn * (3.0 - a * xn * xn);
  temp = xn1 - xn;
  if (temp < 1e-12) {
    cond = 2;
  }
  if (temp > -1e-12) {
    cond = 2;
  }
  xn = xn1;
  res = xn1 * a;
}
```

(a) *Racine carrée par la méthode de Newton*

```
double a = [0.8,0.9];
double b = 0.35;
double x = [0.79,0.99]_[-0.1,0.1];

while (1) {
  x = a * x * x - b * x * x;
}
```

(b) *Suite hypergéométrique*

Figure 14. *Code source des exemples*

sion est une implantation logicielle de l'arithmétique flottante permettant de fixer arbitrairement le nombre de bits de la mantisse. Bien que l'interprétation abstraite permette de calculer des approximations des comportements de programmes, il est, en général, nécessaire d'utiliser des stratégies d'itérations permettant en un temps fini de calculer le point fixe avec suffisamment de précision. Dans nos exemples la stratégie employée est le dépliage de boucle (Blanchet *et al.*, 2003), c'est-à-dire l'exécution du corps de la boucle un certain nombre de fois fixé au début de l'analyse.

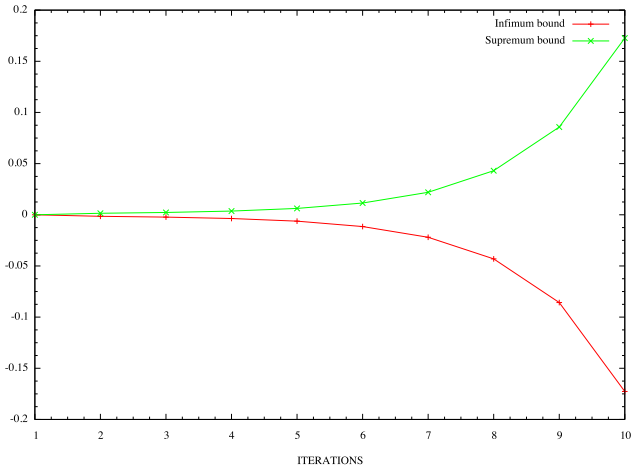
De plus, nous avons appliqué, pour le premier exemple, un opérateur d'élargissement, quasi indispensable en interprétation abstraite, mais difficile à définir en analyse de la précision numérique. En effet, une méthode courante pour définir cet opérateur utilise un ensemble de valeurs paliers (Blanchet *et al.*, 2003). Dans notre cas, il n'est pas possible de définir a priori des paliers servant pour l'élargissement des termes d'erreur. La technique alors adoptée est la dégradation de la précision. A chaque application de l'opérateur d'élargissement, nous diminuons le nombre de bits servant à représenter les erreurs. Cette diminution a pour effet d'augmenter la valeur de l'*ulp* et permet ainsi d'avoir un ensemble dynamique de paliers.

Le premier programme implante une méthode de Newton calculant la racine carrée d'un nombre a strictement positif ; le code source est donné à la figure 14(a). Ce calcul s'appuie sur la récurrence définie par l'équation :

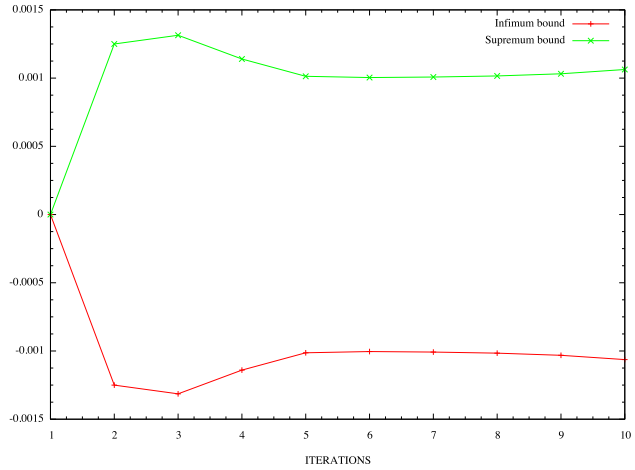
$$x_{n+1} = \frac{x_n}{2}\left(3 - a x_n^2\right) \ .$$

Le résultat du programme, c'est-à-dire la racine carrée r d'un nombre a, est $r = x_p \times a$ avec x_p la valeur du point fixe de la fonction. Cet algorithme nécessite une connaissance d'une "bonne valeur" initiale proche de la solution, nous ne considérons

pas ce problème ici en supposant cette valeur connue. Nous avons analysé les propriétés numériques de ce programme avec les sémantiques $[\![.]\!]^\sharp_{\mathbb{E}}$ et $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$ et un dépliage initial de la boucle de 10 itérations. Notre sémantique de l'erreur différentiée permet d'améliorer le calcul des erreurs en prenant en compte les relations qui les lient. Mais elle n'agit pas sur les valeurs flottantes. Afin d'assurer la stabilité de l'algorithme dans les flottants, nous avons choisi de calculer la racine carrée d'une valeur simple 25.0 mais entachée d'une erreur $[-0.01, 0.01]$.



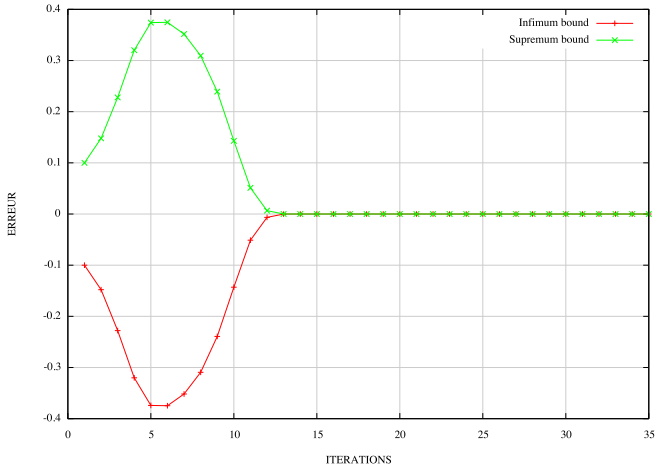
(a) *Erreur calculée par la sémantique* $[\![.]\!]^\sharp_{\mathbb{E}}$



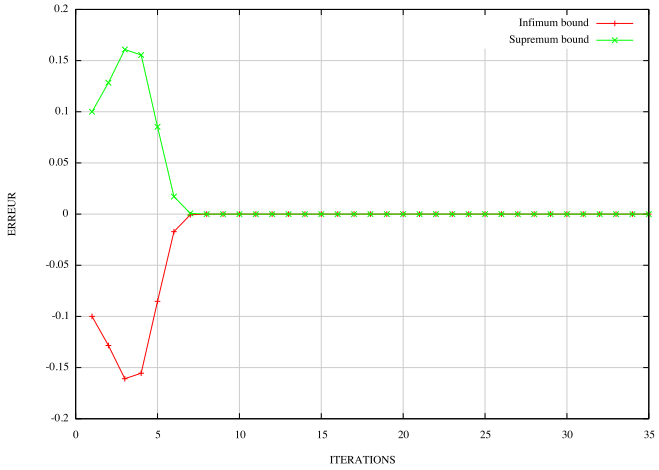
(b) *Erreur calculée par la sémantique* $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$

Figure 15. *Intervalle d'erreurs pour chaque itération du programme 14(a)*

L'analyse avec la sémantique $[\![.]\!]^{\sharp}_{\mathbb{E}}$ qui utilise l'arithmétique d'intervalles pour le calcul du flottant et de l'erreur ne permet pas d'apprécier la qualité du résultat flottant qui est $[5.0, 5.0]$ alors que l'évolution de l'intervalle d'erreur grandit. La figure 15(a) montre cette évolution en fonction des itérations. La sémantique $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$ permet, grâce à la prise en compte des relations entre erreurs, de calculer un terme d'erreur qui converge. L'évolution de l'erreur calculée par approximation linéaire est donnée à la



(a) *Erreur calculée par la sémantique* $[\![.]\!]^{1\sharp}_{\mathbb{ED}}$



(b) *Erreur calculée par la sémantique* $[\![.]\!]^{2\sharp}_{\mathbb{ED}}$

Figure 16. *Intervalle d'erreurs pour chaque itération du programme 14(b)*

figure 15(b). Grâce à ces informations nous pouvons conclure que le résultat flottant est très proche du résultat réel.

Le second exemple est $x_{n+1} = ax_n^2 - bx_n^2$. Le programme associé à cette suite est donné à la figure 14(b). Nous avons comme valeur initiale x_0 l'intervalle flottant $[0.79, 0.99]$ et celle-ci est entachée d'une erreur comprise dans l'intervalle $[-0.1, 0.1]$, c'est-à-dire que les valeurs réelles sont dans l'intervalle $[0.69, 1.09]$. La constante a est dans l'intervalle $[0.8, 0.9]$ et la constante b est égale à 0.35. Naturellement, cette suite converge positivement vers 0 quand n tend vers $+\infty$.

Une analyse par la sémantique $[\![.]\!]_{\mathbb{E}}^{\sharp}$ conduit à une valeur flottante qui converge vers 0 mais un terme d'erreur qui rapidement explose (à partir de la 13-ième itération) pour être de la forme $[-\text{MAX\_FLOAT}, \text{MAX\_FLOAT}]$ avec $\text{MAX\_FLOAT} = 1.7976931348623157e^{308}$. Ce résultat est uniquement dû à l'arithmétique d'intervalles qui ne permet pas de détecter que la soustraction utilise le même x_n^2.

Par contre, une analyse par la sémantique $[\![.]\!]_{\mathbb{E}}^{1\sharp}$ permet de calculer un terme d'erreur plus proche du modèle mathématique. L'évolution de l'erreur est donnée à la figure 16(a). Nous obtenons alors un terme d'erreur convergeant vers 0 au bout d'une douzaine d'itérations avec une amplitude de $[-0.3742, 0.3742]$. Le passage à l'ordre 2 dans le calcul de l'erreur en utilisant la sémantique $[\![.]\!]_{\mathbb{E}}^{2\sharp}$ permet d'affiner encore ce résultat comme le montre l'évolution de l'erreur donnée à la figure 16(b). Cette sémantique montre une convergence vers 0 en 7 itérations et avec un intervalle d'erreurs qui est au maximum de $[-0.161, 0.161]$.

L'élévation au carré des x_n engendre des erreurs d'ordre supérieur qui sont de plus en plus prépondérantes dans le calcul de l'erreur globale. L'application directe de l'arithmétique d'intervalles ne permet pas de détecter les relations entre les erreurs et donc fournit un mauvais résultat. La différentiation au premier ordre capte les relations entre erreurs élémentaires mais pas entre erreurs d'ordre supérieur ce qui conduit à un bon résultat mais qui peut encore être amélioré, ce que fait la sémantique à l'ordre deux.

5. Conclusion

Dans cet article, nous avons présenté une nouvelle arithmétique dédiée à l'étude de la précision numérique dans le cadre de la validation de programmes numériques. Cette arithmétique est issue de la combinaison de deux méthodes numériques existantes : l'erreur globale et la différentiation automatique. Elle permet de calculer plus finement les erreurs d'arrondi en diminuant l'effet enveloppant dans les calculs et ainsi permet d'obtenir des analyses plus précises. Nous avons montré que dans certains cas la méthode de l'erreur globale différentiée à l'ordre 2 donne des résultats plus précis qu'au premier ordre. Il serait aussi intéressant d'étudier les ordres supérieurs. Cependant, il est nécessaire de tenir compte du compromis entre efficacité et gain de précision, surtout qu'il n'est pas évident que les ordres supérieurs améliorent les résultats si significativement qu'ils justifient tous ces calculs.

Dans cet article, nous nous sommes concentrés sur l'amélioration du calcul des erreurs d'arrondi. Or les termes d'erreurs dépendent aussi des valeurs flottantes, en particulier dans le cas de la multiplication. Une sur-approximation de cet ensemble de flottants se répercute dans le calcul de l'erreur même avec la technique présentée ici. Une solution envisagée est d'utiliser des formes relationnelles qui introduisent des contraintes linéaires pour affiner le calcul d'ensemble de valeurs flottantes (Goubault *et al.*, 2005; Goubault *et al.*, 2006a) et ainsi de réduire l'effet enveloppant au niveau de la partie flottante. (Goubault *et al.*, 2006a) définissent une sémantique relationnelle fondée sur des formes affines permettant de limiter l'effet enveloppant dans le calcul flottant mais pas dans le calcul des erreurs. Une combinaison entre cette sémantique et la sémantique de l'erreur différentiée conduirait à des analyses très précises prenant en considération les dépendances entre flottants et erreurs. (Goubault *et al.*, 2005) proposent aussi de prendre en compte toutes ces dépendances, en ajoutant à la sémantique de (Goubault *et al.*, 2006a) des formes affines entre les erreurs. Une comparaison de cette sémantique complètement relationnelle avec la combinaison de la sémantique de (Goubault *et al.*, 2006a) et de la nôtre est envisagée.

La différentiation automatique est une technique qui a été très étudiée (Griewank, 2000; Hascoët *et al.*, 2005; Bischof *et al.*, 1997; Bischof *et al.*, 1994). Il serait très intéressant d'un point de vue optimisation d'étudier encore plus attentivement les moyens mis en œuvre pour calculer les dérivées afin d'améliorer le temps de calculs de l'analyse statique présentée dans cet article. Une piste à explorer serait l'utilisation de résultats issus d'analyses statiques préparatoires comme par exemple (Tadjouddine *et al.*, 1998) afin de réduire le nombre de calculs ou l'espace mémoire.

Par ailleurs, la connaissance des dérivées de la fonction $\mathcal{E}$, nous laisse entrevoir la possibilité de définir un opérateur d'élargissement adapté pour l'étude de la précision numérique. L'opérateur d'élargissement, élément important de la théorie de l'interprétation abstraite, permet d'accélérer la convergence du calcul de point fixe. Nous examinons la possibilité de définir un tel opérateur en se fondant sur l'évolution des dérivées. Excepté la possible phase d'initialisation, les variables d'une boucle d'un programme suivent, en général, une progression comparable à celle définie par une suite mathématique. Une piste envisageable serait alors l'utilisation du domaine suites arithmético-géométriques (Feret, 2005) afin d'estimer la progression des dérivées.

6. Bibliographie

Ait-Ameur Y., « Refinement of Rational End-Points Real Numbers by Means of Floating-Point Numbers », *Science of Computer Programming*, vol. 33, n° 2, p. 133-162, 1999.

Bajard J.-C., Beaumont O., Chesneaux J.-M., Daumas M., Erhel J., Michelucci D., Muller J.-M., Philippe B., Revol N., Roch J.-L., Vignes J., *Qualité des Calculs sur Ordinateurs. Vers des arithmétiques plus fiables ?*, Masson, 1997.

Bischof C. H., Carle A., Khademi P. M., Mauer A., The ADIFOR 2.0 System for the Automatic Differentiation of Fortran 77 Programs, Technical Report n° MCS–P481–1194, Argonne, Ill., 1994.

Bischof C. H., Roh L., Mauer-Oats A. J., « ADIC : an Extensible Automatic Differentiation Tool for ANSI-C », *Software Practice and Experience*, vol. 27, n° 12, p. 1427-1456, 1997.

Bischof C., Hovland P., Norris B., « Implementation of Automatic Differentiation Tools », *Partial Evaluation and Semantics-Based Program Manipulation (PEPM'02)*, ACM Press, New York, NY, USA, p. 98-107, 2002.

Blanchet B., Cousot P., Cousot R., Feret J., Mauborgne L., Miné A., Monniaux D., Rival X., « A Static Analyzer for Large Safety-Critical Software », *Programming Language Design and Implementation (PLDI'03)*, ACM, ACM Press, p. 196-207, 2003.

Chesneaux J.-M., *L'arithmétique Stochastique et le Logiciel CADNA*, Université Pierre et Marie Curie (Paris 6), 1995. Habilitation à diriger des recherches.

Chesneaux J., Vignes J., « Sur la robustesse de la méthode CESTAC », *Comptes Rendus de l'Académie des Sciences, Paris*, vol. 307, n° 1, p. 855-860, 1988.

Cousot P., « Interprétation abstraite », *Technique et science informatique*, vol. 19, n° 1-2-3, p. 155-164, January, 2000.

Cousot P., Cousot R., « Abstract Interpretation : a Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints », *Principles of Programming Languages (POPL'77)*, ACM, ACM Press, p. 238-252, 1977.

Cousot P., Cousot R., « Abstract Interpretation Frameworks », *Journal of Logic and Symbolic Computation*, vol. 2, n° 4, p. 511-547, 1992.

Feret J., « The Arithmetic-Geometric Progression Abstract Domain », *Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, n° 3385 in *LNCS*, Springer Verlag, p. 42-58, 2005.

Goldberg D., « What Every Computer Scientist Should Know About Floating-Point Arithmetic », *ACM Computing Surveys*, vol. 23, n° 1, p. 5-48, 1991.

Goubault E., « Static Analyses of Floating-Point Operations », *Static Analysis Symposium (SAS '01)*, vol. 2126 of *LNCS*, Springer Verlag, 2001.

Goubault E., Putot S., « Weakly Relational Domains for Floating-Point Computation Analysis », *First International Workshop on Numerical and Symbolic Abstract Domains*, 2005.

Goubault E., Putot S., « Static Analysis of Numerical Algorithms », *Static Analysis Symposium (SAS '06)*, LNCS, Springer Verlag, 2006a.

Goubault E., Putot S., Martel M., « Some Future Challenges in the Validation of Control Systems », *Embedded Real Time Software (ERTS'06)*, SIA, 2006b.

Griewank A., *Evaluating Derivatives : Principles and Techniques of Algorithmic Differentiation*, Society for Industrial and Applied Mathematics, 2000.

Hascoët L., Araya-Polo M., « The Adjoint Data-Flow Analyses : Formalization, Properties, and Applications », *Automatic Differentiation : Applications, Theory, and Tools (AD'04)*, LNCS, Springer, 2005.

IEEE Task P754, *ANSI/IEEE 754-1985, Standard for Binary Floating-Point Arithmetic*, IEEE, New York. 1985.

Jaulin L., Kieffer M., Didrit O., Walter E., *Applied Interval Analysis*, Springer, 2001.

Langlois P., Automatic Linear Correction of Rounding Errors, Technical report, INRIA, 1999.

Martel M., « An Overview of Semantics for the Validation of Numerical Programs », *Verification, Model Checking and Abstract Interpretation (VMCAI'05)*, vol. 3385 of *LNCS*, Springer Verlag, 2005.

Martel M., « Semantics of Roundoff Error Propagation in Finite Precision Calculations », *Journal of Higher Order and Symbolic Computation*, vol. 19, p. 7-30, 2006.

Monniaux D., « The Pitfalls of Verifying Floating-Point Computations », *ACM TOPLAS*, 2007. To appear.

Moore R., *Methods and Applications of Interval Arithmetic*, Studies in Applied Mathematics, SIAM, 1979.

Neumaier A., « Taylor Forms–Use and Limits », *Reliable Computing*, vol. 9, n° 1, p. 43-79, 2003.

Potts P. J., Edalat A., Escardó H. M., « Semantics of Exact Real Arithmetic », *Procs of Logic in Computer Science*, IEEE Computer Society Press, 1997.

Revol N., Introduction to Interval Arithmetic, Technical Report n° RR2001-41, LIP, École Normale Supérieure de Lyon and INRIA, 2001.

Rudin W., *Principles of Mathematical Analysis*, McGraw-Hill, 1976.

Rump S., « Fast and Parallel Interval Arithmetic », *BIT Numerical Mathematic*, vol. 39, n° 3, p. 539-560, 1999.

Tadjouddine M., Eyssette F., Faure C., « Sparse Jacobian Computation in Automatic Differentiation by Static Program Analysis », *Static Analysis Symposium (SAS '98)*, vol. 1503 of *LNCS*, p. 311-326, 1998.

Vignes J., « A Survey of the CESTAC Method », *Real Numbers and Computer Conference*, 1996.

Article reçu le 22 octobre 2007
Accepté après révisions le 22 mai 2008

Alexandre Chapoutot *est doctorant en informatique. Il est actuellement en troisième année de thèse au Commissariat à l'Energie Atomique (CEA-LIST) au sein du laboratoire Modélisation et Analyse des Systèmes en Interaction (MeASI). Ses travaux portent sur la validation de propriétés numériques de modèles Simulink fondée sur les méthodes d'analyse statique par interprétation abstraite.*

Matthieu Martel *est maître de conférence habilité à diriger des recherches à l'Université de Perpignan depuis septembre 2007. Auparavant, il a été chercheur au Commissariat à l'Energie Atomique (CEA-LIST) pendant 7 ans et chargé d'enseignements à temps incomplet à l'Ecole Polytechnique pendant 5 ans. Au CEA, Matthieu Martel a participé au développement théorique et pratique du logiciel Fluctuat, un interpréteur abstrait pour la précision numérique développé en collaboration avec de nombreux partenaires industriels dans le domaine de l'aéronautique. A l'Ecole Polytechnique, il a participé à la mise en place et au développement de la Chaire de Systèmes Complexes financée par le groupe Thales et notamment à la création du Master d'Ingénierie des Systèmes Industriels Complexes. Au laboratoire ELIAUS de l'Université de Perpignan, Matthieu Martel est membre de l'équipe de recherche DALI, spécialisée dans l'arithmétique des ordinateurs. Il travaille actuellement sur des techniques d'interprétation abstraite pour améliorer automatiquement l'implémentation des calculs effectués sur ordinateurs.*