Vers un système de contraintes pour l'analyse des erreurs de précision des calculs sur les flottants*

Remy Garcia[1†] **Claude Michel**[1] **Marie Pelleau**[1] **Michel Rueher**[1]

[1] Université Côte d'Azur, CNRS, I3S, France
{prénom.nom}@i3s.unice.fr

Résumé

Les calculs sur les nombres flottants induisent des erreurs liées aux arrondis nécessaires à la clôture de l'ensemble des flottants. Ces erreurs, symptomatiques de la distance entre le calcul sur les flottants et le calcul sur les réels, sont à l'origine de nombreux problèmes, tels que la précision ou la stabilité de ces calculs. Elles font l'objet de nombreux travaux qui s'appuient sur une surestimation des erreurs effectives. Si ces approches permettent une estimation de l'erreur, estimation qui peut être affinée en découpant l'espace de recherche en sous-domaines, elles ne permettent pas, à proprement parler, de raisonner sur ces erreurs et, par exemple, de produire un jeu de valeurs d'entrée capable d'exercer une erreur donnée. Afin de pallier ce manque et d'enrichir les possibilités d'analyses de ces erreurs de précision, nous introduisons dans un solveur de contraintes sur les flottants un domaine dual à celui des valeurs, le domaine des erreurs. Ce domaine est associé à chacune des variables du problème. Nous introduisons aussi les fonctions de projections qui permettent le filtrage de ces domaines ainsi que les mécanismes nécessaires à l'analyse de ces erreurs.

Abstract

Floating-point computations induce errors linked to the rounding operations required to close the set of floating-point numbers. These errors, which are symptomatic of the distance between the computations over the floats and the computation over the real numbers, are at the origin of many problems, such as the precision or the stability of floating-point computations. They are the subject of numerous works which are based on an overestimation of actual errors. These approaches allow an estimate of the error, an estimate that can be refined by splitting the search space into subdomains, but they do not, strictly speaking, make it possible to reason about these errors and, for example, to produce input values that excercise a given error. In order to overcome this lack and to enrich the possibilities of analysis of these errors, we introduce in a solver for constraints over the floats, a domain dual to that of the values, the domain of the errors. This domain is associated with each of the variables of the problem. We also introduce the projection functions that allow the filtering of these domains as well as the mechanisms required for the analysis of these errors.

1 Introduction

Les calculs sur les nombres flottants induisent des erreurs liées aux arrondis nécessaires à la clôture de l'ensemble des flottants. Ces erreurs, symptomatiques de la distance entre le calcul sur les flottants et le calcul sur les réels, sont à l'origine de nombreux problèmes, tels que la précision ou la stabilité de ces calculs. Elles sont d'autant plus critiques que l'utilisateur de ces calculs fait le plus souvent abstraction de la nature particulière de l'arithmétique des flottants et les considère comme des calculs sur les réels. Identifier, quantifier et localiser ces erreurs est une tâche ardue qui ne peut, le plus souvent, qu'être accomplie à l'aide d'outils qui l'automatisent.

Un exemple bien connu de déviation du calcul liée aux erreurs de calcul sur les flottants est le polynôme de Rump [14] :

$$333.75b^6 + a^2(11a^2b^2 - b^6 - 121b^4 - 2) + 5.5b^8 + a/(2b)$$

avec $a = 77617$ et $b = 33096$. La valeur exacte de cette expression, calculée en utilisant la bibliothèque de calcul GMP [7], est $-54767/66192 \approx -0.827396056$. Lorsque cette expression est évaluée avec des flottants simples et un arrondi au plus près, le résultat obtenu est alors $\approx 6.3382530011411 \times 10^{29}$, soit

*Ces travaux ont été partiellement supportés par l'ANR COVERIF (ANR-15-CE25-0002).

†Papier doctorant : Remy Garcia[1] est auteur principal.

une valeur très éloignée de la valeur réelle. L'importance de la différence entre ces deux calculs, qui est de $\approx -6.3382530011411 \times 10^{29}$, souligne l'intérêt d'outils d'analyse des erreurs d'arrondi.

Les erreurs de calcul sur les flottants font l'objet de nombreux travaux qui s'appuient sur une surestimation des erreurs effectives. Ainsi l'interprète abstrait Fluctuat [6, 5] combine l'arithmétique affine et les zonotopes afin d'analyser la robustesse des programmes sur les flottants. PRECiSA [15, 13] se base sur une analyse statique du programme pour évaluer les erreurs d'arrondi. Les travaux sur l'amélioration automatique du code de Nasrine Damouche [3], ainsi que ceux d'Eva Darulova [4], s'appuient sur une évaluation de l'erreur d'arrondi afin d'estimer la distance entre l'expression sur les flottant et l'expression sur les réels. Si ces approches permettent une estimation de l'erreur, estimation qui peut être affinée en découpant l'espace de recherche en sous-domaines, elles ne permettent pas, à proprement parler, de raisonner sur ces erreurs et, par exemple, de produire un jeu de valeurs d'entrée capable d'exercer une erreur donnée.

Afin de pallier ce manque et d'enrichir les possibilités d'analyses de ces erreurs de précision, nous introduisons dans un solveur de contraintes sur les flottants [16, 10, 1, 11, 12] un domaine dual à celui des valeurs, le domaine des erreurs. Ce domaine est associé à chacune des variables du problème. Nous introduisons aussi les fonctions de projections qui permettent le filtrage de ces domaines ainsi que les mécanismes nécessaires à l'analyse de ces erreurs.

Plus précisément, nous nous positionnons sur l'analyse de la déviation des calculs en flottants par rapport aux réels. En particulier, nous ignorons volontairement la possibilité d'une erreur physique initiale sur les données d'entrées. Et pour des raisons de simplicité, nous nous restreignons aux quatre opérations arithmétiques de base. Cette simplification permet aussi un calcul exact des valeurs sur les réels [1], tant des valeurs des expressions que des erreurs de calcul. Enfin, le mode d'arrondi est supposé être le mode d'arrondi par défaut, i.e., l'arrondi au plus près pair.

2 Notation et définitions

L'ensemble des nombres flottants, un sous-ensemble fini des rationnels, a été introduit pour approximer les nombres réels sur un ordinateur. La norme IEEE pour les nombres flottants [9] définie le format des différents types de flottants ainsi que le comportement des opérations arithmétique sur ces nombres. Dans la suite, les flottants sont restreint au plus courant, i.e. les flottants binaires en simple précision sur 32 bits et les flottants binaires en double précision sur 64 bits.

Un nombre flottant binaire v est représenté par un triplet (s, e, m) où s est le signe de v, e son exposant et m sa mantisse. Quand $e > 0$, v est normalisé et sa valeur est donnée par :

$$(-1)^s \times 1.m \times 2^{e-bias}$$

où le *bias* permet de représenter les valeurs négatives de l'exposant. Par exemple, pour les flottants 32 bits, s vaut 1 bit, e vaut 8 bits, m vaut 23 bits et *bias* est égal à 127.

x^+ désigne le plus petit nombre flottant strictement plus grand que x tandis que x^- désigne le plus grand nombre flottant strictement plus petit que x. C'est-à-dire que x^+ est le successeur de x alors que x^- est son prédécesseur.

Un *ulp*, pour *unit in the last place*, est la distance qui sépare deux flottants consécutifs. Cependant, cette définition est ambiguë pour les flottants qui sont des puissances de 2, tel que 1.0 : dans ce cas, et si $x > 0$, alors $x^+ - x = 2 \times (x - x^-)$. À chaque fois que cela sera nécessaire, une formulation explicite de la distance sera utilisée.

3 Quantification de la déviation du calcul

Les calculs sur les flottants se distinguent des calculs sur les réels par l'utilisation d'arrondis. L'ensemble des nombres flottants étant un sous-ensemble fini des nombres réels, le résultat d'une opération sur les flottants n'est pas, en général, un nombre flottant. Afin de clore l'ensemble des nombres flottants pour ces opérations, ce résultat doit être arrondi au nombre flottant le plus proche selon une direction d'arrondi préalablement choisie.

La norme IEEE 754 [9] définit le comportement de l'arithmétique des flottants. Pour les quatre opérations de base, cette norme impose un arrondi *correct* : le résultat d'une opération sur les flottants doit être égal à l'arrondi du résultat de l'opération équivalente sur les réels. On a alors $z = x \odot y = round(x \cdot y)$ où z, x et y sont des nombres flottants, $\odot$, l'une des quatre opérations arithmétiques de base sur les flottants, à savoir, $\oplus$, $\ominus$, $\otimes$ et $\oslash$, $\cdot$ étant l'opération équivalente sur les réels et, *round*, la fonction d'arrondi. Cette propriété limite l'erreur introduite par une opération sur les flottants à $\pm\frac{1}{2}Ulp(z)$ pour les opérations correctement arrondies au plus proche pair, l'arrondi le plus courant.

Dès lors que le résultat d'une opération sur les flottants est arrondi, ce résultat est différent du résultat attendu sur les réels. Et chacune des opérations d'une

1. En utilisant une bibliothèque de calcul sur les rationnels et aux limitations de mémoire près.

$$Addition : z = x \oplus y \rightarrow e_z = e_x + e_y + e_{\oplus}$$
$$Soustraction : z = x \ominus y \rightarrow e_z = e_x - e_y + e_{\ominus}$$
$$Produit : z = x \otimes y \rightarrow e_z = x_{\mathbb{F}} e_y + y_{\mathbb{F}} e_x + e_x e_y + e_{\otimes}$$
$$Division : z = x \oslash y \rightarrow e_z = \frac{y_{\mathbb{F}} e_x - x_{\mathbb{F}} e_y}{y_{\mathbb{F}}(y_{\mathbb{F}} + e_y)} + e_{\oslash}$$

FIGURE 1 – Calcul de la déviation pour les opérations de base

expression complexe est susceptible d'introduire une différence entre le résultat escompté sur les réels et le résultat obtenu sur les flottants. Alors que pour une opération donnée le flottant obtenu est optimal, en termes d'arrondi, le cumul de ces approximations peut conduire à des déviations importantes comme dans le cas du polynôme de Rump.

L'origine de la déviation d'un calcul sur les flottants par rapport à son équivalent sur les réels se situant au niveau de chaque opération élémentaire, il est possible de reconstruire cette déviation à partir de la composition du comportement de chaque opération élémentaire. Considérons l'une de ces opérations, la soustraction. Si les variables d'entrée de ces opérations résultent d'un calcul, elles sont entachées d'une erreur. Par exemple, pour la variable x, la déviation sur le calcul de x, e_x est donnée par $e_x = x_{\mathbb{R}} - x_{\mathbb{F}}$ où $x_{\mathbb{R}}$ dénote le résultat attendu sur les réels et $x_{\mathbb{F}}$ dénote le résultat obtenu sur les flottants. Notez qu'à la différence d'une erreur physique, e_x est signé. Ce choix est rendu nécessaire pour capturer correctement des comportements spécifiques aux flottants comme les compensations des erreurs qui peuvent intervenir au sein d'un calcul sur les flottants.
La déviation du calcul due à la soustraction peut alors être calculée de la manière suivante : pour $z = x \ominus y$, l'erreur sur z, e_z est égale à $(x_{\mathbb{R}} - y_{\mathbb{R}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}})$. Puisque $e_x = x_{\mathbb{R}} - x_{\mathbb{F}}$ et $e_y = y_{\mathbb{R}} - y_{\mathbb{F}}$, on a

$$e_z = ((x_{\mathbb{F}} + e_x) - (y_{\mathbb{F}} + e_y)) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}})$$

La déviation entre le résultat sur les réels et le résultat sur les flottants pour une soustraction peut donc se calculer grâce à la formule suivante :

$$e_z = e_x - e_y + ((x_{\mathbb{F}} - y_{\mathbb{F}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}}))$$

Dans cette formule, le terme $((x_{\mathbb{F}} - y_{\mathbb{F}}) - (x_{\mathbb{F}} \ominus y_{\mathbb{F}}))$ caractérise l'erreur commise par l'opération de soustraction, que nous noterons $e_{\ominus}$. La formule se simplifie alors en :

$$e_z = e_x - e_y + e_{\ominus}$$

Elle comporte deux éléments : d'une part la combinaison des déviations qui entachent les valeurs d'entrée et, d'autre part, la déviation introduite par l'opération élémentaire.

La figure 1 formule la déviation du calcul pour les quatre opérations de base. Pour chacune de ces formules, le calcul de l'erreur combine les déviations qui entachent les valeurs d'entrée avec l'erreur introduite par l'opération courante. On peut aussi observer que pour le produit et la division cette déviation est proportionnelle aux valeurs d'entrée.

L'ensemble de ces formules permet le calcul de la différence entre le résultat attendu sur les réels et celui obtenu sur les flottants pour une expression complexe. C'est à partir d'elles qu'est bâti notre solveur de contraintes sur les erreurs sur les flottants.

4 Domaines d'erreurs

Dans un *CSP* classique, à toute variable x est associé $\mathcal{D}_{\mathbf{x}}$ son domaine de valeurs. Celui-ci dénote l'ensemble des valeurs possibles que cette variable peut prendre. Dans le cas des nombres flottants, le domaine des valeurs est représenté par un intervalle de flottants à bornes dans $\mathbb{F}$:

$$\mathbf{x}_{\mathbb{F}} = [\underline{x}_{\mathbb{F}}, \overline{x}_{\mathbb{F}}] = \{x_{\mathbb{F}} \in \mathbb{F}, \underline{x}_{\mathbb{F}} \leq x_{\mathbb{F}} \leq \overline{x}_{\mathbb{F}}\}$$

avec $\underline{x}_{\mathbb{F}} \in \mathbb{F}$ et $\overline{x}_{\mathbb{F}} \in \mathbb{F}$.

Les erreurs de calcul constituent une autre dimension à prendre en compte. Elles nécessitent un domaine spécifique de par la nature distincte des éléments à représenter, mais aussi, des valeurs possibles des erreurs qui appartiennent à l'ensemble des réels. Un domaine des erreurs est donc introduit ; domaine associé à chaque variable du problème. Et, puisque les contraintes traitées ici sont réduites aux quatre opérations de base, et que ces quatre opérations sont appliquées à des flottants, i.e., un sous-ensemble fini des rationnels, ce domaine peut être représenté par un intervalle de rationnels à bornes dans $\mathbb{Q}$:

$$\mathbf{e}_x = [\underline{e}_x, \overline{e}_x] = \{e_x \in \mathbb{Q}, \underline{e}_x \leq e_x \leq \overline{e}_x\}$$

avec $\underline{e}_x \in \mathbb{Q}$ et $\overline{e}_x \in \mathbb{Q}$.

Un autre domaine d'erreur est nécessaire au bon fonctionnement de notre système, le domaine des erreurs des opérations $e_{\odot}$. Contrairement aux précédents

domaines, il n'est pas lié à chaque variable du problème mais à chaque *instance* d'une opération arithmétique du problème. Tout comme le domaine des erreurs attaché à une variable, il prend ses valeurs dans l'ensemble des rationnels. On aura donc :

$$\mathbf{e}_\odot = [\underline{e}_\odot, \overline{e}_\odot] = \{e_\odot \in \mathbb{Q}, \underline{e}_\odot \leq e_\odot \leq \overline{e}_\odot\}$$

avec $\underline{e}_\odot \in \mathbb{Q}$ et $\overline{e}_\odot \in \mathbb{Q}$.

Ce triptyque domaine de valeurs, domaine d'erreurs et domaine d'erreurs sur les opérations est nécessaire pour représenter l'ensemble des phénomènes liés d'une part aux valeurs que les variables peuvent prendre et d'autre part aux différentes erreurs qui entrent en jeu dans les calculs sur les flottants.

5 Fonctions de projection

Le processus de filtrage de notre solveur s'appuie sur des fonctions de projections classiques pour réduire les domaines des variables. Si les domaines de valeurs utilisent les fonctions de projections définies dans [11] et étendues dans [1] et [10], les domaines d'erreurs nécessitent leurs propres fonctions de projections.

Les projections des domaines d'erreurs sont obtenues par extension aux intervalles des formules de la figure 1. Ces formules étant sur les réels, elles ne présentent aucune difficulté particulière lors de leur extension aux intervalles. Par exemple, pour la soustraction, on obtient les quatre fonctions de projection suivantes :

$$\mathbf{e}_z \leftarrow \mathbf{e}_z \cap (\mathbf{e}_x - \mathbf{e}_y + \mathbf{e}_\ominus)$$

$$\mathbf{e}_x \leftarrow \mathbf{e}_x \cap (\mathbf{e}_z + \mathbf{e}_y - \mathbf{e}_\ominus)$$

$$\mathbf{e}_y \leftarrow \mathbf{e}_y \cap (-\mathbf{e}_z + \mathbf{e}_x + \mathbf{e}_\ominus)$$

$$\mathbf{e}_\ominus \leftarrow \mathbf{e}_\ominus \cap (\mathbf{e}_z - \mathbf{e}_x + \mathbf{e}_y)$$

où $\mathbf{e}_x$, $\mathbf{e}_y$ et $\mathbf{e}_z$ sont les domaines d'erreurs des variables x, y et z, d'une part, et $\mathbf{e}_\ominus$ est le domaine d'erreurs de la soustraction, d'autre part.

Les fonctions de projections sur les domaines des erreurs ne portent que sur les opérations arithmétiques et l'affectation qui transmet l'erreur de calcul de l'expression à la variable affectée. Les comparateurs n'effectuent des projections que sur les domaines de valeurs car l'erreur n'intervient pas dans les comparaisons.

L'ensemble de ces fonctions de projections sont utilisées pour réduire les différents domaines des variables jusqu'à atteindre un point fixe. Pour des raisons d'efficacité, mais aussi pour pallier la potentielle présence de convergences lentes, le calcul du point fixe est arrêté dès lors qu'aucune réduction de domaine n'est supérieure à 5%.

6 Liens entre domaine de valeurs et d'erreurs

Afin de permettre à l'un des domaines, qu'il soit de valeurs ou d'erreur, de bénéficier des réductions de l'autre domaine, il faut établir des relations entre ces deux domaines.

La première de ces relations, et sans doute la plus importante, lie le domaine des valeurs avec celui des erreurs sur les opérations. Elle provient de la garantie offerte par la norme IEEE 754 que les opérations arithmétiques de base sont correctement arrondies. Les quatre opérations de base étant correctement arrondies vers le plus proche flottant pair, on a :

$$(x \odot y) - ((x \odot y) - (x \odot y)^-)/2 \leq (x \cdot y)$$

$$(x \cdot y) \leq (x \odot y) + ((x \odot y)^+ - (x \odot y))/2$$

où x^- et x^+ sont, respectivement, le flottant qui précède, succède, à x. Autrement dit, le résultat sur les flottants est à un demi ulp, la distance qui sépare deux flottants successifs, du résultat sur les réels. Et l'erreur sur l'opération est donc contenue dans cet ulp :

$$-((x \odot y) - (x \odot y)^-)/2 \leq e_\odot \leq +((x \odot y)^+ - (x \odot y))/2$$

Cette équation établit une relation entre le domaine des valeurs et celui des erreurs sur les opérations : cette erreur ne peut pas être plus grande que le plus grand demi ulp du domaine de valeurs du résultat de l'opération. La projection du domaine de valeurs du résultat de l'opération sur l'erreur sur l'opération est obtenue en étendant cette formule aux intervalles :

$$\mathbf{e}_\odot \leftarrow \mathbf{e}_\odot \cap [-min((\underline{z} - \underline{z}^-), (\overline{z} - \overline{z}^-))/2,$$

$$+max((\underline{z}^+ - \underline{z}), (\overline{z}^+ - \overline{z}))/2]$$

Notez que la contraposée de cette propriété offre une seconde opportunité de lier domaine des valeurs et domaines des erreurs. En effet, puisque l'erreur sur une opération est inférieure en valeur absolue au plus grand demi ulp du domaine des valeurs du résultat de l'opération, alors si $|e_\odot| \geq \delta > 0$, les plus petites valeurs du domaine résultat de l'opération en question ne peuvent être support d'une solution. Pour ces petites valeurs, en valeur absolue proche de zéro, si leur demi ulp est plus petit que δ, elles ne peuvent être associées à une erreur sur l'opération assez grande pour être dans le domaine de $e_\odot$.

7 Contraintes sur les erreurs

Les contraintes habituellement disponibles dans un solveur de contraintes établissent des relations entre

les variables du problème. La dualité des domaines disponibles dans notre solveur requiert d'introduire une distinction entre domaine de valeurs et domaine d'erreurs. Afin de préserver la sémantique courante des expressions, les variables continuent de dénoter les valeurs possibles. C'est sous la forme d'une fonction dédiée, $err(x)$, qu'il est possible d'exprimer des contraintes sur les erreurs. Par exemple, $abs(err(x)) \geq \epsilon$ dénote une contrainte qui impose que l'erreur sur la variable x est, en valeur absolue, supérieure à ϵ. Notez que les erreurs prenant leurs valeurs dans $\mathbb{Q}$, la contrainte porte sur les rationnels.

Lorsqu'une contrainte mêle erreurs et variables, les variables, dont les domaines sont des flottants, sont promues en rationnels. La contrainte est alors une contrainte sur les rationnels.

8 Exemples

8.1 Implémentation

Le domaine d'erreurs ainsi que les fonctions de projections on été implémenté dans Objective-CP [8], un outil d'optimisation proposant plusieurs solveur, dont un de programmation par contraintes. Ce solveur supporte déjà les contraintes sur les flottants, à travers FPCS [12](Floating Point Constraint Solver). Les opérations sur les rationnels, par exemple dans les fonctions de projections des erreurs, sont réalisées à l'aide de la librairie GMP (GNU Multiple Precision Arithmetic Library) [7].

8.2 Le polynôme de Rump

Le premier exemple que nous proposons est le polynôme de Rump présenté en introduction. Sur ce problème, le solveur n'a qu'à propager les valeurs d'entrée de a et b pour obtenir la valeur du polynôme (dans la variable r). Le solveur affiche alors les éléments suivants :

```
a : 7.76170000e+04 (YES)
ea: [0.00000000e+00, 0.00000000e+00]
b : 3.30960000e+04 (YES)
eb: [0.00000000e+00, 0.00000000e+00]
r : 6.33825300e+29 (YES)
er: [-6.33825300e+29, -6.33825300e+29]
```

qui correspondent bien aux valeurs attendues. Cet exemple montre le fonctionnement correct des fonctions de projections sur les erreurs.

8.3 Second exemple

Ce second exemple porte sur le calcul de l'expression $z = (x + y) - (x/y)$ avec $x = 0.1$ et $y \in [0.2, 0.4]$, avec, pour x et y, des erreurs initiales à 0. y étant un intervalle, le filtrage ne peut instancier les valeurs des différentes variables.

```
float x = 0.1f;
float y = [0.2f, 0.4f];
float z;
z = (x + y) - (x/y);
```

Objective-CP réduit le domaine de z à $[-1.99999988e{-}1, 2.50000000e{-}1]$ et son domaine d'erreur à $[-7.45058060e{-}9, 1.49011612e{-}8]$. Pour Fluctuat, le domaine de z est réduit à $[-1.99999988e{-}1, 2.50000000e{-}1]$, soit le même intervalle que celui proposé par notre solveur. Par contre, le domaine de l'erreur sur z n'est réduit qu'à $[-5.96046448e{-}8, 5.96046448e{-}8]$, soit un intervalle plus grand que dans notre cas.

On peut supposer que cette différence est principalement liée à la soustraction pour laquelle l'erreur est traitée comme une différence dans notre cas alors qu'en terme d'erreur physique, les erreurs sont ajoutées.

8.4 Solve cubic

La fonction solve cubic calcule les racines réelles de l'expression $x^3 + ax^2 + bx + c = 0$. Cette version a été prise dans la GSL (GNU Scientific Library) [2], une bibliothèque largement utilisée, qui propose de nombreux outils et fonctions dédiés au calcul scientifique. Le corps de la fonction `solve_cubic` est écrit de la façon suivante :

```
int gsl_poly_solve_cubic (double a,
                          double b, double c, ...) {
  double q = (a * a - 3 * b);
  double r = (2 * a * a * a - 9 * a * b + 27 * c);

  double Q = q / 9;
  double R = r / 54;

  double Q3 = Q * Q * Q;
  double R2 = R * R;

  double CR2 = 729 * r * r;
  double CQ3 = 2916 * q * q * q;

  if (R == 0 && Q == 0) {
    ...
  } else if (CR2 == CQ3) {
    ...
  } else if (R2 < Q3) {
    ...
  } else {
    ...
  }
}
```

À la lecture de ce code, une question se pose sur la première condition, i.e. `R == 0 && Q == 0` : existe-t-il des valeurs d'entrée pour lesquelles R et Q sont

égal à zéro et s'il y en a, est-ce que R et Q sont calculés avec une erreur. Effectivement, il n'est pas recommandé de tester si une variable est égale à zéro dans les programmes numériques (hormis pour éviter une exception, comme une division par zéro). Afin de répondre à cette question, considérons les intervalles suivant pour les variables d'entrées, $a \in [14.0, 16.0]$, $b \in [-200, 200]$, et $c \in [-200, 200]$, de plus, considérons que ces variables ne sont pas entachées d'erreur, i.e., que l'erreur de calcul initiale sur ces variables est nulle. R et Q dépendent respectivement de r et q qui eux même dépendent directement des variables d'entrées. Ainsi, un CSP qui représente toutes ces relations des variables d'entrées jusqu'à la condition étudiée est :

$$
\begin{aligned}
&a \in [14.0, 16.0] \wedge b \in [-200, 200] \wedge c \in [-200, 200] \wedge \\
&err(a) = 0 \wedge err(b) = 0 \wedge err(c) = 0 \wedge \\
&q = (a * a - 3 * b) \wedge \\
&r = (2 * a * a * a - 9 * a * b + 27 * c) \wedge \\
&Q = q/9 \wedge \\
&R = r/54 \wedge \\
&R == 0 \wedge Q == 0
\end{aligned}
$$

Ces contraintes sont suffisantes pour s'assurer que le programme atteigne bien la première branche if. Cependant, ce CSP ne pose pas encore de contraintes sur l'erreur de R et de Q.

Dans un premier temps, il est intéressant de savoir si il existe, au sein de leurs domaines respectifs, des valeurs des variables d'entrée a, b, et c telles que R et Q sont exactement égal à 0. À ces fins, il faut ajouter les contraintes $err(Q) = 0$ et $err(R) = 0$ dans le CSP. Comme résultat, le solveur écrit que lorsque a est fixé à 15, b est fixé à 75, et c est fixé à 125, les erreurs sur Q et R sont égales à 0. Il est intéressant de noter que toutes les autres variables du programme sont mises à 0 avec une erreur à 0.

Dans un second temps, soulevons la question de l'existence de valeurs d'entrées pour lesquelles la condition est vraie alors que R et Q sont calculés avec des erreurs. Pour cela, il suffit de remplacer les contraintes sur les erreurs par $err(Q) > 0$ et $err(R) > 0$. Comme avec les contraintes précédentes, ce modèle est résolu dans Objective-CP et donne $a = 1.50100000e{+}01$, $b = 7.51000333e{+}01$, et $c = 1.25250167e{+}02$ comme solution. Avec ces valeurs d'entrées, Q et R sont toujours égals à 0 mais avec une erreur de $8.14913569e{-}16$ et $1.30826243e{-}14$ respectivement. De plus, les autres variables sont aussi égales à 0 mais avec une erreur supérieure à 0.

Par conséquent, légerement modifier les valeurs d'entrées d'une expression permet de passer d'un calcul correct à un calcul entaché d'erreurs d'arrondis. De plus, à cause de l'arrondi, une condition peut être vraie sur les flottants, mais fausse sur les réels.

9 Conclusion

Dans cet article, nous avons introduit un solveur de contraintes capable de raisonner sur les erreurs de calcul sur les flottants. Il repose sur un système de domaines duals, le premier représentant les valeurs possibles qu'une variable du problème peut prendre, le second représentant les erreurs commises lors du calcul de ces erreurs. Des domaines particuliers attachés aux instances des opérations arithmétiques des expressions numériques utilisées dans les différentes contraintes du problème représentent les erreurs dues à chacune de ces opérations. Augmenté des fonctions de projections sur les erreurs et de contraintes sur les erreurs, notre solveur offre des possibilités uniques de raisonner sur les erreurs de calcul.

La prochaine étape de nos travaux consiste à améliorer la recherche de solutions en présence de contraintes sur les erreurs et à ajouter des capacités d'optimisation globale pour, par exemple, déterminer pour quel jeu de valeurs d'entrée, l'erreur est maximale.

Références

[1] Bernard Botella, Arnaud Gotlieb, and Claude Michel. Symbolic execution of floating-point computations. *Software Testing, Verification and Reliability*, 16(2) :97–121, 2006.

[2] GSL Project Contributors. GSL - GNU scientific library - GNU project - free software foundation (FSF). http ://www.gnu.org/software/gsl/, 2010.

[3] Nasrine Damouche, Matthieu Martel, and Alexandre Chapoutot. Improving the numerical accuracy of programs by automatic transformation. *STTT*, 19(4) :427–448, 2017.

[4] Eva Darulova and Viktor Kuncak. Sound compilation of reals. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 235–248. ACM, 2014.

[5] Khalil Ghorbal, Eric Goubault, and Sylvie Putot. A logical product approach to zonotope intersection. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 212–226, 2010.

[6] Eric Goubault and Sylvie Putot. Static analysis of numerical algorithms. In *Static Analysis, 13th International Symposium, SAS 2006, Seoul, Korea, August 29-31, 2006, Proceedings*, volume 4134 of *Lecture Notes in Computer Science*, pages 18–34, 2006.

[7] Torbjörn Granlund and the GMP development team. GNU MP : The GNU Multiple Precision Arithmetic Library, 2016. `http://gmplib.org/`.

[8] Pascal Van Hentenryck and Laurent Michel. The objective-cp optimization system. In *19th International Conference on Principles and Practice of Constraint Programming (CP 2013)*, pages 8–29, 2013.

[9] IEEE. *754-2008 - IEEE Standard for floating point arithmethic*, 2008.

[10] Bruno Marre and Claude Michel. Improving the floating point addition and subtraction constraints. In *Proceedings of the 16th international conference on Principles and practice of constraint programming (CP'10)*, LNCS 6308, pages 360–367, St. Andrews, Scotland, 6–10th September 2010.

[11] Claude Michel. Exact projection functions for floating point number constraints. In *AI&M 1-2002, Seventh international symposium on Artificial Intelligence and Mathematics (7th ISAIM)*, Fort Lauderdale, Floride (US), 2–4th January 2002.

[12] Claude Michel, Michel Rueher, and Yahia Lebbah. Solving constraints over floating-point numbers. In *7th International Conference on Principles and Practice of Constraint Programming (CP 2001)*, pages 524–538, 2001.

[13] Mariano Moscato, Laura Titolo, Aaron Dutle, and César A. Muñoz. Automatic estimation of verified floating-point round-off errors via static analysis. In Stefano Tonetta, Erwin Schoitsch, and Friedemann Bitsch, editors, *Computer Safety, Reliability, and Security : 36th International Conference, SAFECOMP 2017, Trento, Italy, September 13-15, 2017, Proceedings*, volume 10488 of *Lecture Notes in Computer Science*, pages 213–229, Cham, 2017. Springer International Publishing.

[14] Siegfried M. Rump. Algorithms for verified inclusions : Theory and practice. In Ramon E. Moore, editor, *Reliability in Computing : The Role of Interval Methods in Scientific Computing*, pages 109–126. Academic Press Professional, Inc., San Diego, CA, USA, 1988.

[15] Laura Titolo, Marco Feliu, Mariano Moscato, and César Muñoz. An abstract interpretation framework for the round-off error analysis of floating-point programs. In Isil Dillig and Jens Palsberg, editors, *Proceedings of the 19th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI 2018)*, volume 10747 of *Lecture Notes in Computer Science*, pages 516–537, Los Angeles, CA, UAS, January 2018. Springer.

[16] Heytem Zitoun, Claude Michel, Michel Rueher, and Laurent Michel. Search strategies for floating point constraint systems. In *23rd International Conference on Principles and Practice of Constraint Programming, CP 2017*, LNCS 10416, pages 707–722, Melbourne, Autralia, 28–1st August 2017.