Techniques pour améliorer la précision numérique des algorithmes d'optimisation

Romain Megel

LocalSolver, 36 avenue Hoche, Paris, France
rmegel@localsolver.com

Mots-clés : *précision numérique, nombre flottant*

1 Introduction

La majeure partie des techniques de résolution mises en oeuvre dans les projets d'optimisation aujourd'hui font intervenir des nombres flottants. D'usage très répandu dans de nombreuses disciplines scientifiques, l'arithmétique flottante a l'avantage d'être très peu coûteuse tant en mémoire qu'en temps dans le matériel informatique usuel. Malheureusement, les flottants sont aussi la source de nombreux maux pour les praticiens de la recherche opérationnelle : l'utilisation d'algorithmes itératifs ou incrémentaux tels que ceux utilisés classiquement en recherche locale ou dans les solveurs de programmation linéaire (simplexe, points intérieurs) ont la facheuse tendance à dériver numériquement.

Dès lors deux questions fondamentales se posent : comment limiter les dérives numériques de nos algorithmes sans sacrifier la performance ? et comment garantir la qualité des résultats retournés par nos algorithmes et nos outils ?

L'exposé reposera sur le retour d'expérience acquis dans le développement du solveur de programmation mathématique LocalSolver qui est un solveur d'optimisation de type *model & run* [1] et qui hybride plusieurs techniques de résolution dont la recherche locale.

2 Limiter les dérives numériques

Dans cet exposé, nous rappellerons en préambule la structure des nombres flottants, leurs propriétés et regarderons les instructions mises à notre disposition par les fondeurs de microprocesseurs pour les manipuler. A cette occasion, nous présenterons quelques méthodes simples pour identifier et anticiper toute perte de précision numérique lors des calculs. Nous ferons ensuite un panorama de quelques algorithmes classiques utilisables par tous pour améliorer la précision des calculs sur des opérations n-aire comme la somme ou le produit scalaire en évoquant l'algorithme de sommation de Kahan [2] ou la sommation en cascade (*pairwise summation*).

Une autre manière de limiter les dérives numériques consiste à utiliser des nombres de plus grande précision, comme les flottants de type quad-precision (binary128). Malheureusement, ces derniers n'étant pas nativement supportés par les microprocesseurs actuels, ils sont très souvent implémentés de manière logiciel, occassionant d'énormes facteurs de ralentissement. Nous verrons que d'autres techniques sont possibles, notamment la technique dite du double-double [3]. Cette dernière consiste à représenter un nombre flottant comme la somme de deux nombres flottants au format binary64. Sous certaines conditions, l'usage du double-double apporte une précision quasi équivalente au format binary128 tout en étant beaucoup plus rapide car reposant sur une implémentation purement matérielle.

L'ensemble des techniques présentées ont été implémentées au sein du solveur LocalSolver et nous ont permis de réduire considérablement les problèmes de dérives numériques sur de nombreux cas pathologiques tout en ayant un impact très limité sur les performances.

3 Garantir la qualité des résultats

Une manière usuelle de fournir des garanties sur les résultats retournés par un algorithme basé sur des nombres flottants est d'utiliser des techniques issues de l'arithmétique d'intervalle [4]. L'arithmétique d'intervalle consiste à retourner un encadrement contenant de manière certaine le résultat d'un calcul. Formellement, il s'agit d'étendre toute fonction $f : \mathbb{R} \mapsto \mathbb{R}$ en une fonction $f' : \mathbb{R}^2 \mapsto \mathbb{R}^2$ pour laquelle $\forall x \in [a, b], f(x) \in f'([a, b])$.

Nous verrons comment implémenter de manière efficace une telle arithmétique sur les microprocesseurs à jeu d'instruction les plus répandus aujourd'hui (x86, amd64) et comment elle est actuellement utilisée dans le presolve de LocalSolver pour simplifier et transformer les modèles des utilisateurs.

Références

[1] F. Gardi, T. Benoist, J. Darlay, B. Estellon, et R. Megel. *Mathematical Programming Solver Based on Local Search*. Wiley, 2014.

[2] W Kahan. Further remarks on reducing truncation errors, commun. Assoc. Comput. Mach, 8 :40, 1965

[3] D.E. Knuth The Art of Computer Programming (2nd ed.). chapter 4.2.3. problem 9.

[4] R.E. Moore Methods and applications of interval analysis. SIAM Studies in Applied Mathematics, 1979.