



# Queste de savoir

Quand la calculatrice nous trompe

---

1<sup>er</sup> août 2022



# Table des matières

	Introduction . . . . .	1
1.	Programme simple . . . . .	2
2.	Identification de l'erreur . . . . .	3
	2.1. Cas problématique . . . . .	3
	2.2. Source du problème . . . . .	3
	2.3. Cause du problème . . . . .	4
3.	Résolution de l'erreur . . . . .	4
	Conclusion . . . . .	6
4.	Références pour aller plus loin . . . . .	7

## Introduction

«Vous n'aurez pas toujours une calculatrice dans la poche!» Cette phrase, prononcée par tant de prof de maths, tombe en désuétude à l'heure où les téléphones glissés dans nos poches sont parfois plus puissants que les supercalculateurs d'il y a 30 ans. Mais à quel point pouvons-nous faire confiance aux machines à calculer pour fournir les bons résultats?

Les ordinateurs actuels, qui sont tout simplement de grosses machines à calculer, enregistrent généralement les nombres sur 64 bits, ce qui permet de représenter une quinzaine de chiffres décimaux. On aimerait qu'après quelques calculs avec ces nombres, le résultat bénéficie également d'une telle précision. C'est souvent le cas, mais pas toujours.

En effet, des calculs simples peuvent donner des résultats faux dès les premiers chiffres, malgré la quinzaine de chiffres stockés par la machine! Ces cas sont relativement rares, mais lorsqu'ils concernent la trajectoire d'une fusée ou la résistance d'un gratte-ciel, cela peut porter à conséquence. Surtout que ce type d'erreurs est dur à détecter, le résultat du calcul étant, *a priori*, inconnu.

**Cet article montre comment il est simple de se laisser tromper par un programme de calcul en apparence juste.** Il prend pour exemple un programme de résolution des équations du second degré, qui présente (au moins) une erreur discrète affectant sensiblement les résultats. Sa correction n'est pas triviale, et montre que les calculs faits par ordinateur sans précaution ne sont pas aussi infaillibles que l'on pourrait le penser au premier abord.

Si vous êtes familier avec les équations du second degré et que vous avez déjà touché une calculatrice dans votre vie, vous êtes prêts pour une virée au pays parfois surprenant du calcul numérique. C'est parti!

## 1. Programme simple

Par définition, une équation du second degré peut toujours s'écrire de la forme suivante:

$$ax^2 + bx + c = 0$$

avec  $a, b, c$  trois nombres réels et  $a$  non-nul.

Pour la résoudre, rien de plus simple que de partir de la méthode classique, telle qu'enseignée au lycée. Elle consiste à:

1. Calculer le discriminant:  $\Delta = b^2 - 4ac$
2. Calculer les solutions en fonction de la valeur du discriminant  $\Delta$ .
  - S'il est négatif, il n'y a pas de solution réelle.
  - S'il est nul, l'unique solution est  $x_1 = -\frac{b}{2a}$ .
  - S'il est positif, les deux solutions sont  $x_1 = \frac{-b + \sqrt{\Delta}}{2a}$  et  $x_2 = \frac{-b - \sqrt{\Delta}}{2a}$ .

Voici un programme de résolution des équations du second degré utilisant cette méthode, écrit en Python.

```
1 from math import sqrt
2
3 # Définition des coefficients
4 a = 1.0 # doit être non-nul !
5 b = -4.0
6 c = 3.0
7
8 # Résolution de l'équation
9 delta = b * b - 4 * a * c
10 if delta < 0: # pas de solution réelle
11     solutions = []
12 elif delta == 0: # une seule solution
13     solutions = [-b / (2 * a)]
14 elif delta > 0: # deux solutions
15     solutions = [(-b + sqrt(delta)) / (2 * a), (-b - sqrt(delta))
16                 / (2 * a)]
17 else: # impossible
18     print("Euh... Kamoulox ?")
19 # Affichage du résultat
20
21 print("Δ = {}".format(delta))
22 for i, x in enumerate(solutions):
23     print("x_{} = {}".format(i + 1, x))
```

En testant rapidement le programme, on constate qu'il se comporte bien au premier abord. Par exemple pour  $a = 1, b = -4$  et  $c = 3$ , il affiche le résultat correct suivant.

## 2. Identification de l'erreur

```
1 Δ = 4.0
2 x_1 = 3.0
3 x_2 = 1.0
```

## 2. Identification de l'erreur

### 2.1. Cas problématique

Notre petit programme semble fonctionner, mais fonctionne-t-il quelles que soient les valeurs des coefficients  $a$ ,  $b$  et  $c$ ? Regardons en détail le cas  $a = 1$ ,  $b = -10^7$  et  $c = 1$ .

Pour ces valeurs, notre programme renvoie les résultats ci-dessous.

```
1 Δ = 999999999999996.0
2 x_1 = 9999999.9999999
3 x_2 = 9.96515154838562e-08
```

Cependant, les approximations des solutions exactes<sup>1</sup>, sont les suivantes:

$x \approx 9.999999999999... \times 10^6$

$x \approx 1.000000000000... \times 10^{-7}$

On voit que si la première solution semble correcte, la deuxième présente une erreur dès le troisième chiffre. Notre programme fournit donc un résultat faux, alors même qu'un ordinateur peut normalement calculer avec une quinzaine de chiffres. D'où vient l'erreur?

### 2.2. Source du problème

Pour tenter de comprendre où se glisse l'erreur, calculons la solution étape par étape. La première étape consiste à calculer le discriminant.

```
1 >>> b*b -4*a*c
2 999999999999996.0
```

Il s'agit de la valeur exacte, il n'y a pas de problème. Continuons en calculant la racine carrée.

---

1. Obtenues avec [Wolfram Alpha](#), qui utilise des méthodes spécifiques pour calculer avec autant de décimales que souhaitées. On parle de calcul en précision arbitraire.

### 3. Résolution de l'erreur

```
1 >>> math.sqrt(b*b -4*a*c)
2 9999999.9999998
```

Une approximation de la racine carrée exacte donne un résultat similaire, il n'y a pas non plus de problème. Continuons encore.

```
1 >>> -b - math.sqrt(b*b -4*a*c)
2 1.993030309677124e-07
```

Ce n'est pas totalement à côté de la plaque, mais le résultat exact de ce calcul s'arrondit comme suit:

$2.000000000000... \times 10^{-7}$

Le quatrième chiffre de notre calcul est donc faux. Il y a anguille sous roche!

#### 2.3. Cause du problème

Pourquoi n'a-t-on que trois chiffres justes alors qu'on pourrait en espérer au moins une dizaine?

Le problème vient du fait que  $\sqrt{b^2 - 4ac}$  est très proche de  $-b$ , ce qui est désavantageux lors de la soustraction.

Regardons schématiquement ce qu'il se passerait si on stockait exactement 16 chiffres décimaux.

$-b - \sqrt{b^2 - 4ac} = 10,000,000\{, \} \setminus, 000\{, \} \setminus, 00 - 9,999,999\{, \} \setminus, 999\{, \} \setminus, 801 = 0\{, \} \setminus, 000\{, \} \setminus, 199 = 1,99 \times 10^{-7}$

Lors du calcul, de nombreux chiffres s'annulent, et seuls les derniers chiffres apparaissent effectivement dans le résultat final... Cela signifie que les chiffres nécessaires pour un résultat précis sont gâchés par des chiffres sans importance, car commun aux deux termes de la soustraction. Dans un tel cas, au lieu d'exploiter une quinzaine de chiffres, le résultat n'en exploite qu'une poignée (trois dans notre exemple), et est donc très imprécis.

Ce phénomène apparaissant dès que l'on soustrait deux nombres proches s'appelle une *annulation catastrophique* ou *annulation massive*, et nuit gravement à la précision des calculs. Peut-on y remédier dans notre cas?

### 3. Résolution de l'erreur

*signe*

Pour résoudre le problème d'annulation catastrophique, il existe plusieurs voies, parmi lesquelles:

- utiliser un système de calcul formel pour obtenir une solution exacte,
- calculer avec plus de décimales pour mitiger l'annulation catastrophique,

### 3. Résolution de l'erreur

- utiliser des fractions, avec lesquelles on peut calculer de manière exacte,
- contourner l'annulation catastrophique pour éviter de soustraire deux nombres proches.

Les deux premières solutions sont simples, et devraient être utilisées si possible, en particulier par les non-spécialistes. Elles ont cependant le désavantage d'être peu performantes, et donc inadaptées au calcul intensif, mais également de ne pas être utilisables sur de nombreux dispositifs tels que les calculatrices et les calculateurs embarqués<sup>1</sup>.

La solution utilisant les fractions n'est pas toujours adaptée, ni performante d'ailleurs, en particulier quand il s'agit d'effectuer des calculs dont le résultat peut être irrationnel (racine carrée typiquement).

Nous allons donc voir une solution utilisant la quatrième voie, c'est-à-dire qui contourne la soustraction problématique.

En regardant notre problème de près, on remarque que si  $\sqrt{b^2 - 4ac}$  est proche de  $b$ , deux cas apparaissent:

- soit  $b$  est positif et il faut éviter de calculer  $-b + \sqrt{b^2 - 4ac}$ ,
- soit  $b$  est négatif et il faut éviter de calculer  $-b - \sqrt{b^2 - 4ac}$ .

Autrement dit, il faut éviter de calculer la solution suivante directement:

$$x_1 = \frac{-b + \text{signe}(b) \sqrt{b^2 - 4ac}}{2a} \sim,$$

avec

$$gal1pourbpositifounulet - 1sinon.$$

Par contre, rien ne nous empêche de calculer l'autre solution comme précédemment.

$$x_2 = \frac{-b - \text{signe}(b) \sqrt{b^2 - 4ac}}{2a} \sim.$$

Pour calculer  $x_1$  sans utiliser la formule habituelle, on peut utiliser la propriété suivante qui lie les deux solutions de l'équation:

$$x_1 x_2 = \frac{c}{a}$$

On en déduit la formule suivante, qui n'a plus d'annulation catastrophique.

$$x_1 = \frac{c}{ax_2} = \frac{2c}{-b - \text{signe}(b) \sqrt{b^2 - 4ac}}$$

En mettant à jour le programme initial avec notre astuce, on obtient la version améliorée ci-dessous.

```
1 from math import sqrt
2
3 # Définition des coefficients
4 a = 1
5 b = -1e7
6 c = 1
7
8 # Résolution de l'équation
```

---

1. Par exemple, les très populaires cartes Arduino ne sont pas adaptées au calcul formel et sont limitées à 32 bits pour la représentation des nombres.

## Conclusion

```
9 delta = b * b - 4 * a * c
10 if delta < 0: # pas de solution réelle
11     solutions = []
12 elif delta == 0: # une seule solution
13     solutions = [-b / (2 * a)]
14 elif delta > 0: # deux solutions
15     signe_b = 1 if b >= 0 else -1
16     solutions = [(2 * c) / (-b - signe_b * sqrt(delta)), (-b -
17                 signe_b * sqrt(delta)) / (2 * a)]
18 else: # impossible
19     print("Euh... Kamoulox ?")
20 # Affichage du résultat
21 print("Δ = {}".format(delta))
22 for i, x in enumerate(solutions):
23     print("x_{} = {}".format(i, x))
```

On teste avec les valeurs problématiques de tout à l'heure et on obtient:

```
1 Δ = 999999999999996.0
2 x_1 = 1.000000000000001e-07
3 x_2 = 9999999.9999999
```

Ce qui est très proche des approximations des valeurs exactes:

$$x \approx 1.000000000000001 \times 10^{-7}$$

$$x \approx 9.999999999999999 \times 10^6$$

Problème résolu!

## Conclusion

L'exemple de cet article, la résolution d'équations du second degré, peut sembler un problème sans grand intérêt, tant il est simple. Pourtant, nous avons vu comment un calcul numérique peu précautionneux peut fournir des résultats surprenants, alors même que le programme de calcul semble en apparence juste.

Le phénomène d'annulation massive nous a causé des problèmes, et nous avons pu l'éviter sans sortir l'artillerie lourde (calcul formel, calcul avec nombre de chiffres augmenté). L'origine de l'annulation massive se trouve dans le nombre restreint de chiffres utilisés pour stocker les nombres. Cette particularité du calcul par ordinateur est la source de nombreux autres problèmes, étudiés par les chercheurs et ingénieurs en analyse numérique.





Alors, peut-on faire confiance à sa calculatrice? Oui, mais si on veut un calcul précis, alors il faut tenir compte des limitations des ordinateurs et être éventuellement astucieux. Il n'existe pas de solution miracle aux problèmes de précision. Si certains conseils pratiques peuvent aider







#### 4. Références pour aller plus loin

(éviter les soustractions quand les deux termes peuvent être proches, par exemple), ils ne sont pas forcément suffisants pour analyser un problème spécifique. Le domaine reste pointu, et les logiciels spécialisés exploitent de nombreuses techniques pour offrir des outils de calcul à la fois précis et performants.

#### 4. Références pour aller plus loin

- [Scilab is not naive \(en anglais\)](#)  explique comment le logiciel de calcul scientifique Scilab calcule intelligemment les solutions aux équations du second degré. Il s'agit d'un approfondissement des problématiques de notre article.
- [Contrôler la propagation des erreurs de calculs numériques](#)  présente la méthode CESTAC, qui vise à estimer la propagation des erreurs d'arrondis.
- [Loss of significance sur Wikipedia \(en anglais\)](#)  développe le même exemple que cet article, mais illustre en plus un cas d'annulation massive dans le calcul du discriminant.
- [Virgule flottante sur Wikipédia](#)  présente en détail la méthode la plus couramment utilisée pour représenter des nombres dans les ordinateurs.

---

Merci à [SpaceFox](#)  ,  , [Karnaj](#)  et [Gabbro](#)  pour leurs retours lors de la bêta, ainsi qu'à [artragis](#)  et [Arius](#)  pour la validation.

*Illustration du tutoriel: Photographie d'une calculette par Adrian Pingstone, 2004, domaine public.*