
Bibliothèque de calcul en arithmétique à virgule fixe



[3]

Étudiants :

BERBAGUI Amine
DELABORDE Romane
PAVEL Dumitru

Enseignant :

Mr HILAIRE Thibault

Projet d'initiation

MAIN 3 2020-2021

27 mai 2021

Table des matières

1	Introduction	2
2	Objectifs	3
3	La virgule fixe	3
4	Fonctionnement	4
5	Précisions sur la bibliothèque	4
6	Création d'une bibliothèque de façon conventionnelle	5
7	Pytest	6
8	La classe FxPVar	7
8.1	Exemples de formats possibles pour la classe FxPVar :	8
8.2	Exemples non valides avec la classe FxPVar :	8
9	La classe FxPConstant	8
10	La classe Adder	8
11	Impacts environnementaux, éthiques et sociétaux	9
12	Bilans personnels :	10
12.1	Amine Berbagui :	10
12.2	Romane Delaborde :	10
12.3	Dumitru Pavel :	10
13	Conclusion	11

1 Introduction

Lorsque l'on parle d'arithmétique en informatique, on parle d'arithmétique à virgule fixe ou d'arithmétique à virgule flottante. L'arithmétique à virgule fixe est une arithmétique permettant des calculs avec des nombres approchant les nombres réels en utilisant uniquement les unités de calcul en entier du processeur. Elle est utilisée dans les systèmes nécessitant peu de ressources matérielles, comme les systèmes embarqués.

Ainsi, une représentation d'un nombre en virgule fixe est un type de donnée correspondant à un nombre qui possède (en base deux ou en base dix) un nombre fixe de chiffres après la virgule. Les nombres en virgule fixe sont utiles pour représenter des quantités fractionnaires dans un format utilisant le complément à deux, si le processeur ne possède aucune unité de calcul en virgule flottante.

Contrairement à l'arithmétique à virgule flottante, la virgule fixe demande plus de précaution et de travail de la part du développeur, puisqu'il doit écrire manuellement les opérations à réaliser (vérification du nombre de bits, décalage de mantisse pour aligner les virgules, ...).

L'objectif principal de notre projet d'initiation consiste à mettre en place une bibliothèque de calcul en arithmétique à virgule fixe. Bien que cette bibliothèque peut s'écrire en C++ (POO), le langage utilisé sera Python pour sa simplicité d'utilisation.

En plus de découvrir ce sujet initialement inconnu de notre groupe, le but sera de rendre cette bibliothèque open-source, documentée en anglais afin de permettre à un grand public de l'utiliser et/ou de la modifier. Par conséquent, il sera nécessaire de présenter notre bibliothèque convenablement, tant sur le fond que sur la forme (PEP 8, ...).

2 Objectifs

Notre but étant d'écrire une librairie permettant l'utilisation de formats variés, les étapes à suivre de notre projet sont les suivantes :

- Définir les opérateurs permettant de modifier un format virgule fixe (en fonction du mode d'arrondi et du mode d'overflow).
- Appliquer les opérations élémentaires dessus : addition, soustraction, multiplication puis éventuellement la division, la racine carré qui sont un peu plus subtiles.
- Cette librairie étant open-source, savoir la documenter correctement et la rendre utilisable par tous.
- Adopter de bonnes pratiques de développement logiciel avec des tests unitaires, documentation et intégration continue.

Éventuellement, nous pourrions rajouter si possible le calcul de l'erreur associé aux opérations réalisées entre deux nombres à virgule fixe.

3 La virgule fixe

Notre projet aborde le thème de la virgule fixe et non de la virgule flottante qui est souvent manipulée dans les ordinateurs actuels. En effet, l'arithmétique flottante peut ne pas toujours être utilisée, par choix ou par incapacité. Il n'est effectivement pas possible d'effectuer de calculs flottants s'il n'existe pas d'unités correspondantes dans l'élément étudié, par exemple dans les micro-contrôleurs.

De plus, l'arithmétique fixe est moins gourmande en ressources et requiert également des circuits de taille moindre comparé à ceux mis en place dans le cadre de l'arithmétique flottante. Par ailleurs, il est préférable d'avoir recours à une bibliothèque de calcul en arithmétique à virgule fixe afin de pouvoir faire du "custom", dans l'optique d'utiliser de diverses tailles de largeur, c'est-à-dire que l'on peut personnaliser les modes d'arrondis, les conversions, etc ...

4 Fonctionnement

Le principe de la virgule fixe reprend celui de la virgule flottante, c'est-à-dire qu'on approche des nombres réels x comme suit : $x = M \times 2^e$ [2]

Avec les deux entiers :

- **M** la mantisse, un entier relatif
- **e** l'exposant, un entier inférieur à 0

Cependant pour la virgule fixe, l'exposant e est implicite et non stocké en mémoire. Par conséquent, tous les calculs réalisés s'effectuent uniquement sur des entiers.

En outre, il est important de ne pas oublier de réaliser des décalages pour aligner les exposants afin de réaliser des calculs entre deux nombres à virgule fixe. Dans les cas contraire, les calculs sont erronés.

5 Précisions sur la bibliothèque

La librairie sur laquelle nous travaillons se démarquera de celles déjà disponibles sur internet.

En effet, ces dernières ne permettent généralement pas de simuler de façon exacte le calcul à virgule fixe. Les erreurs peuvent être dues au fait que ces bibliothèques ne prennent pas en compte les différents modes d'arrondis ou bien que les opérations soient définies implicitement dans le programme.

Notre but est donc de mettre en place une librairie avec divers modes d'arrondis et dont les opérations sont explicitement définies.

Pour qu'elle puisse être accessible à plusieurs personnes, nous déposons tous nos codes sur un serveur Gitlab du laboratoire LIP6^[1] ainsi que les spécifications et autres documents utilisés.

6 Création d'une bibliothèque de façon conventionnelle

Étant donné que notre objectif consiste à mettre en place une bibliothèque libre d'accès, il est nécessaire de la présenter correctement afin que n'importe quel utilisateur puisse la manipuler sans encombre.

Par conséquent, l'ensemble des documents/ressources disponibles sur notre gitlab sont rédigés en anglais afin que le plus grand nombre ait accès à nos travaux.

De plus, tous nos commentaires de code sont également en anglais.

D'autre part, il est important d'organiser son environnement de travail (dans notre cas le gitlab), autant pour notre équipe que pour les futurs utilisateurs de notre librairie.

Il s'agit de la raison pour laquelle nous avons tenté de structurer notre gitlab convenablement.

En outre, nous n'avons exploité qu'une seule branche appelée branche "master", où il est possible de retrouver nos différentes classes ainsi que notre README.

Enfin, il a été nécessaire/préférable de respecter les conventions établies par la PEP 8.

En effet, la PEP 8 est un ensemble de règles qui permet d'homogénéiser le code et d'appliquer de bonnes pratiques. Elle a donc pour objectif de définir des règles de développement communes entre développeurs.

Il est possible de retrouver l'ensemble de ces normes sur le site officiel du PEP 8 qui se **trouve à cette adresse**.

Cependant, dans ce qui suit nous allons tout de même vous présenter les principales conventions à connaître afin de respecter la PEP 8.

- ◆ Les imports sont à placer au début d'un script.
- ◆ Une ligne ne peut pas contenir plus de 80 caractères.
- ◆ Une indentation doit correspondre à 4 espaces.
- ◆ Les noms (variable, fonction, classe, ...) ne doivent contenir que des chiffres ou des lettres sans accent.
- ◆ Pas d'espace avant ":" mais un après.
 - ❖ **Exemple :**
`{test: 2}`
 - ❖ **Contre-exemples :**
`{test : 2}`
`{test:2}`
`{test :2}`
- ◆ Aucun espace avant et après un signe "=" lorsque vous assignez la valeur par défaut du paramètre d'une fonction.

❖ **Exemple :**

```
test=True
```

❖ **Contre-exemples :**

```
test = True
```

```
test =True
```

```
test= True
```

- ◆ Une instruction par ligne.

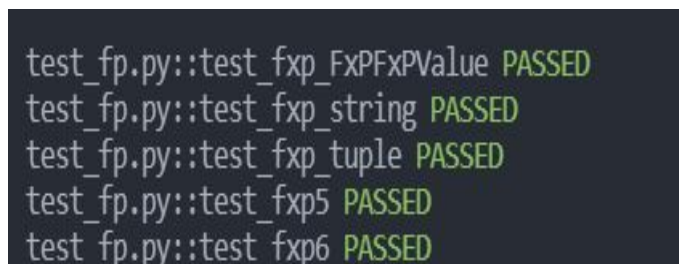
Remarque : Il est possible de vérifier que l'on a bien respecté les normes imposées par le PEP 8 avec par exemple **le site suivant**.

7 Pytest

Dans le but d'adopter les bonnes pratiques de développement mais aussi être certains que toutes les méthodes ou fonctions que nous créons sont correctes, on a mis en place des tests unitaires à l'aide de la librairie **Pytest**. Pour éviter de retaper tous ces tests à chaque modification du code (ce qui arrive souvent lorsqu'un algorithme ou une application est utilisée longtemps) ou à chaque découverte de bugs, ils sont conservés dans un fichier à part. Ceci nous permettra d'exécuter ces tests beaucoup de fois et d'être sûrs que tous les tests seront exécutés. Ces tests sont dit **unitaires** et sont essentiels dans toutes les pratiques courantes de code.

Voici ci-dessous un exemple de test. On veut ici vérifier qu'avec un certain format d'entrée, notre classe **FxPVar** fonctionne correctement et nous renvoie bien la valeur correspondante, ici 2048.

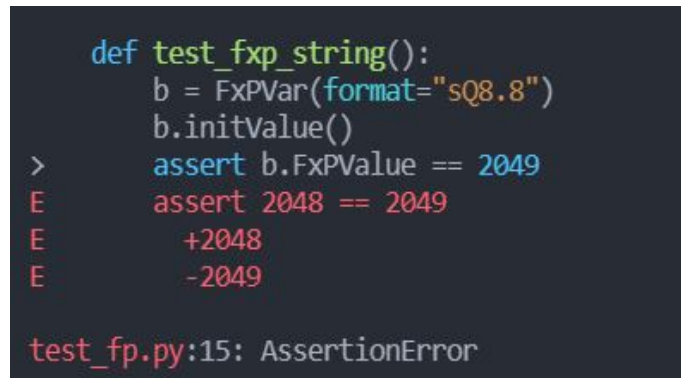
```
1 def test_fxp_string():
2     b = FxPVar(format="sQ8.8")
3     b.initValue()
4     assert b.FxPValue == 2048
```



```
test_fp.py::test_fxp_FxPFxPValue PASSED
test_fp.py::test_fxp_string PASSED
test_fp.py::test_fxp_tuple PASSED
test_fp.py::test_fxp5 PASSED
test_fp.py::test_fxp6 PASSED
```

FIGURE 2: Exemple de test valide avec Pytest

Si maintenant, on s’amuse à mettre une assertion fausse, on a bien une erreur.



```
def test_fxp_string():
    b = FxPVar(format="sQ8.8")
    b.initValue()
> assert b.FxPValue == 2049
E      assert 2048 == 2049
E          +2048
E          -2049
test_fp.py:15: AssertionError
```

FIGURE 3: Exemple de test invalide avec Pytest

8 La classe FxPVar

Notre bibliothèque comporte une classe nommée **FxPVar**. Au lieu de manipuler des **float** (réels à virgule) ou des **int** (entiers relatifs), l'utilisateur va manipuler des **FxPVar**, soit les nombres **msb** et **lsb** (dont les significations seront données plus tard) qui seront donnés dans trois formats différents :

- sous forme de **tuple** : (**msb**,**lsb**)
- sous forme d'une chaîne de caractères. Ce format est également appelé la "Q-notation" : "**sQmsb.lsb**" si le nombre est signé, "**uQmsb.lsb**" sinon
- en initialisant les deux valeurs **msb** (most significant bit) et **lsb** (least significant bit)

Les nombres **FxPVar** auront d'autres caractéristiques telles qu'un entier **wl** de "word length" qui va désigner le nombre de bits nécessaire pour coder ledit nombre, un booléen **signedness** qui va indiquer si le nombre est signé ou non et un flottant **value** qui représente la valeur du nombre. Tous ces paramètres pourront être donnés dans le constructeur directement ou alors assignés individuellement avec des fonctions.

Les attributs de la classe peuvent être accédés par le biais de propriétés **@property**, les attributs de la classe étant protégés

L'addition et la multiplication sont surchargées : on utilisera la classe **Adder** pour effectuer les additions et les soustractions, et la classe **Multiplier** (cf. plus loin, **sections 9 et 10**) pour effectuer les multiplications.

8.1 Exemples de formats possibles pour la classe FxPVar :

```
1 x1 = FxPVar(wl=9, lsb=-4)
2 x2 = FxPVar(format="sQ7.7")
3 x3 = FxPVar(format=(5, -3))
```

8.2 Exemples non valides avec la classe FxPVar :

```
1 x4 = FxPVar(wl=-9, lsb=-4)
```

Le format de x4 est valide mais $wl < 0$ ce qui n'est pas possible.

```
1 x5 = FxPVar(format="sq7.7")
```

L'expression "sq7.7" n'est pas valide car il s'agit d'un q minuscule et non d'un q majuscule, donc elle n'est pas reconnue par la classe.

```
1 x6 = FxPVar(format=(2, -4))
```

Le format de x6 est valide mais on a $msb \leq |lsb|$ donc l'exemple n'est pas validé.

9 La classe FxPConstant

Cette classe reprend exactement les mêmes caractéristiques que la classe FxPVar à l'exception du fait que la valeur `value` ne peut pas être changée (d'où le mot **Constant** dans le nom). De même, elle ne peut être assignée que par le biais du constructeur de la classe.

10 La classe Adder

Comme son nom l'indique la classe Adder permet de réaliser une addition entre deux nombres au format virgule fixe. Étant donné que la classe FxPVar permet de vérifier en amont les formats et donc de s'assurer qu'il n'y a pas de problème lors de la déclaration de chaque FxP, il ne reste plus qu'à réaliser l'opération souhaitée.

Cependant, les deux nombres à virgule fixe ne seront pas nécessairement sur le même format. Ainsi, dans un premier temps, il faudra vérifier que les formats soient compatibles. Si ce n'est pas le cas, le format du résultat correspondra à celui du nombre à virgule fixe

dont la `msb` est la plus grande. Il sera donc essentiel de modifier le format du second nombre à virgule fixe avant de passer à l'opération d'addition.

Supposons deux nombres A et B au même format virgule fixe, ce qui signifie que leur exposant est égal.

Pour procéder à la somme notée C de A et B , il suffit d'additionner leurs mantisses.

$$A = M_A \times 2^e$$

$$B = M_B \times 2^e$$

$$C = (M_A + M_B) \times 2^e$$

11 Impacts environnementaux, éthiques et sociétaux

Notre projet consiste principalement en la création d'une bibliothèque informatique.

Dans le but de mieux comprendre notre projet et ses quelconques intérêts, voyons dans quelle mesure celui-ci aborde les impacts environnementaux, éthiques ou sociétaux.

D'un point de vue environnemental, bien que notre projet ne soit pas un sujet majeur de l'environnement, on peut tout de même s'interroger sur la consommation en ressources informatiques de notre bibliothèque. Cette dernière s'utilise très simplement en l'important sur notre script python comme une librairie classique. La taille est d'à peine quelques kilos octets.

Pour parler du côté éthique, notre projet ne possède qu'un très faible impact éthique ou moral car il ne concerne que l'utilisateur et la machine. On peut tout de même dire que notre projet est entièrement *open-source* et donc libre d'utilisation de quiconque.

12 Bilans personnels :

12.1 Amine Berbagui :

D'un point de vue personnel, ce sujet qui m'était au départ complètement inconnu et qui me paraissait très abstrait est devenu petit à petit plus intéressant dès lors que j'ai appris comment cela fonctionnait. Car si au départ on ne se rend pas compte de l'intérêt d'un tel sujet, c'est lorsque l'on creuse que l'on remarque que beaucoup de machines seraient défaillantes si elles n'avaient pas d'unité de calcul en virgule fixe ou flottante.

De plus, ce projet m'a appris à adopter certaines pratiques essentielles au développement comme la mise en place d'un suivi de projet sur Gitlab, la mise en place de tests unitaires, ...

12.2 Romane Delaborde :

Dans un premier temps, il est à noter que ce sujet n'était pas mon premier choix.

Cependant en travaillant dessus ces dernières semaines, j'ai appris à l'apprécier même si de nombreux points théoriques étaient difficiles à assimiler. En effet, nous avons passé plus de temps à comprendre le sujet plutôt qu'à réellement programmer.

D'un point de vue technique, j'ai approfondi mes connaissances concernant les classes en Python et j'ai appris à respecter les conventions imposées par la PEP 8.

De plus, j'ai découvert la librairie Pytest même si je ne l'ai pas manipulée personnellement. Concernant les difficultés rencontrées, je dirais qu'il a été assez complexe d'assimiler le sujet, en particulier avec la contrainte du travail à distance. Par ailleurs, les ressources mises à disposition étaient quasiment exclusivement en anglais, ce qui peut être un obstacle lorsque l'on ne saisit déjà pas le sujet en français. Cependant, il faudra s'y habituer dans la suite de nos études.

12.3 Dumitru Pavel :

Grâce à ce projet j'ai pu découvrir des nouveaux outils tels que les linters ou gitlab. De plus, j'ai pu améliorer mes connaissances en python, notamment la programmation orientée objet (OOP) et certaines façons de programmer comme le respect de la PEP8 ou le fait qu'on peut déclarer des paramètres de fonctions en spécifiant le type. Ce projet m'a également permis de découvrir le développement en équipe et le monde de l'open source. Par rapport aux projets auxquels j'ai pu participer avant, celui-ci se démarque également par le fait que notre travail doit être documenté en anglais.

Pour résumer, j'ai globalement compris le principe de la virgule fixe ainsi que son utilité dans le monde actuel. En outre, ce projet nous a permis d'approfondir le cours intitulé « Architecture des ordinateurs » présenté par Mme Braunstein.

13 Conclusion

Désormais que notre équipe maîtrise davantage le principe de la virgule fixe, nous sommes d'autant plus déterminés à travailler sur cette librairie. De plus, nous sommes enthousiastes à l'idée de rendre nos travaux publics étant donné que notre bibliothèque sera open-source. Par conséquent, n'importe qui sera en mesure de se l'approprier et de l'enrichir à sa guise.

Ce projet s'est révélé être très intéressant puisqu'il combine parfaitement, à la fois les mathématiques et l'informatique. Il s'inscrit ainsi parfaitement dans notre formation et saura nous être utile pour nos futurs projets. Par ailleurs, celui-ci nous permettra d'approfondir nos connaissances en Python et de découvrir le monde de l'open-source ainsi que le développement de programmes informatiques en équipe.

Références

- [1] Thibault Hilaire. <https://gitlab.lip6.fr/hilaire/efxplib>.
- [2] Thibault Hilaire. Fxp.
- [3] www.python.org. Python logo. <https://fr.wikipedia.org/wiki/Fichier:Python-logo-notext.svg>.