Arithmétique en virgule flottante et norme IEEE 754

TD n° 4

Compétences

- Plus grand et plus petit flottant de la norme IEEE 754
- Initiation à l'analyse numérique à travers la résolution d'une équation du second degré
- Manipuler les structures de contrôle boucle et condition en Python

1 Les flottants de la norme IEEE 754

1.1 Python et les flottants normalisés double précision (64-bits)

L'ensemble des flottants normalisés représentables avec Python sont définis par l'ensemble $\mathbb{F}(2, 53, -1022, 1023)$ tenant sur 8 octets. Déterminer les valeurs du plus petit et du plus grand flottant positif non nul. Vous pourrez écrire ces nombres sous la forme flottante binaire normalisée ($m \times 2^e$) ou bien avec la représentation IEEE.

Jusqu'à maintenant pour répéter plusieurs fois une instruction nous avons utilisé la structure de contrôle ci-contre. Dans ce cas le bloc d'instructions est répété `N` fois. Ce type de répétition dite **inconditionnelle** impose de connaître la valeur de N. Comme toujours l'indentation permet de spécifier le début et la fin du bloc appartenant à la boucle.

Syntaxe Python

```
for i in range(N):
    Ici démarre le bloc d'instructions
```

Mais il arrive souvent que le nombre d'itérations nécessaire ne soit pas connu à l'avance. Prenons l'exemple du calcul du plus petit flottant positif non nul représentatble avec Python. Pour avoir une estimation de cette valeur je peux écrire l'algorithme suivant :

Algorithme 1

```
1: réel x ⟵ 1
2: tant que (x > 0) faire
3:     x = x / 2
4:     affiche x
5: fin tant que
```

Le bloc d'instructions est répété autant de fois que la condition est vérifiée. En Python ce type de répétition dite **conditionnelle** est introduite par le mot clef `while`

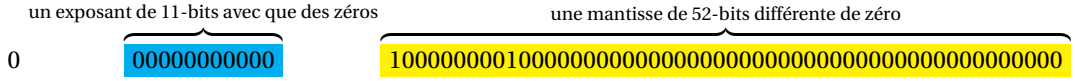
```
while condition:
    bloc d'instructions
```

Ouvrir une console linux (crtl-alt-t) et taper `idle &` Dans le menu `File` de l'IDLE cliquer sur `New Window`. C'est dans cette fenêtre que vous allez écrire votre programme.

1. Rédiger le code python correspondant à *l'algorithme 1*, sauvegarder (ctrl s) sous : `plus_petit64.py` et exécuter (F5).
2. Relever la valeur affichée par le programme.
3. Pourquoi y a-t-il une différence avec la valeur calculée plus haut ? Si vous n'avez pas d'idée passez à la suite...
4. Une des spécification de la norme IEEE est de ne pas renvoyer de message d'erreur dans le cas d'un overflow, les calculs peuvent se poursuivre. La norme prévoit des **nombres spéciaux** ($+inf$ $-inf$ NaN), en guise d'avertisseur dans le cas d'un dépassement, conservant le type *float*. Il est alors possible d'écrire `infini = float('inf')`
   En vous inspirant de *l'algorithme 1* écrire un *algorithme 2* permettant d'avoir une valeur approchée du plus grand flottant normalisé positif en double précision (sauvergarder sous : `plus_grand64.py`)

1.2 Flottants dénormalisés

On rapelle qu'un flottant dénormalisé strictement positif en double précision est de la forme



Le 1 implicite de la normalisation est remplacé par un 0

$$\pm 0, d_1 d_2 d_3 \ldots d_{p-1} \times \beta^{emin}$$

1. Comment représenter la valeur 0 ?
2. Quel est le plus petit flottant positif dénormalisé ? (réponse de l'algorithme 1)

2 La *bonne façon* de résoudre une équation du second degré

Il s'agit de déterminer les solutions réelles de l'équation :

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

Presque tout le monde a déjà écrit sur sa machine à calculer un petit programme permettant de résoudre cette équation.

- Les coefficients a, b, c sont passés en paramètres
- On calcule le descriminant
- On teste le signe du descriminant
- On calcule les solutions

Si on se contente de l'algorithme naturel, les problèmes ont toutes les chances de survenir. Prenons par exemple l'équation suivante

$$0.2013\ x^2 + 16\,777\,216\ x + 0.01234 = 0 \qquad (1)$$

Sur la calculatrice

Une des solutions sur une TI-83 Plus est : 0. Cette solution surprend car 0 n'est pas solution de cette équation.

- Pourquoi la calculatrice renvoie-t-elle 0 (justifier à l'aide d'une phrase) ?
- Vérifier ce résultat sur votre machine à calculer.

Phénomène d'absorption

1. Comment se traduit le phénomène d'absorption dans le calcul du descriminant ? (cf TD_3)
2. Ouvrir à l'aide d'IDLE le programme `trinome.py`. Quel symbole typographique permet d'ajouter des commentaires au programme. Les commentaires permettent de décrire ce que fait le code dans un langage naturel. Ils sont importants pour diverses raisons en voici quelques unes :
   - Expliquer le rôle d'une variable ou d'une fonction
   - Expliquer le fonctionnement d'un algorithme
   - Justifier certains choix techniques
3. Quelles instructions permettent d'intéragir avec l'utilisateur pour la saisie des valeurs a, b, c ?
4. Pour tester le signe du discriminant nous utilisons une instruction de choix introduite par le mot clef `if` Encore une fois il faut bien noter le rôle essentiel de l'*indentation* qui permet de délimiter les blocs d'instructions ainsi que les *deux points* après la condition du choix et après le mot clef `else`

```
if condition:
    bloc d'instructions 1
else:
    bloc d'instructions 2
```

Le terme *condition* désigne une **condition booléenne**, c'est a dire :

— une valeur
— un opérateur de comparaison
— une autre valeur

Les opérateurs de comparaison sont :

==	égal à
<	strictement plus petit que
>	strictement plus grand que
<=	plus petit ou égal
>=	plus grand ou égal
!=	différent de

5. Quelle différence y a-t-il entre `a = 5` et `a == 5` ?
6. Exécuter (F5) le programme `trinome.py` et sasir les valeurs de l'équation (1), commenter.
7. Pour remédier à ce problème on utilise le fait que : $x_1 x_2 = \dfrac{c}{a}$. Dans le cas $\Delta > 0$ il n'y a qu'à calculer l'une des deux racines avec les règles suivantes :
   - Si b > 0, alors parmi les deux racines, il en est une qui est correctement approchée en virgule flottante par $\frac{-b-\sqrt{b^2-4ac}}{2a}$

- Si b < 0, on part de l'expression $\frac{-b+\sqrt{b^2-4ac}}{2a}$ qui conduit à une bonne valeur approchée d'une des deux racines

8. Avant toute chose, commencer par sauvegarder le programme sous `trinome2.py` action indispensable avant la modification d'une version fonctionnelle d'un programme. On se donne ainsi la possibilté de retrouver le source d'origine si les changements apportés au programme ne sont pas satisfaisants.
   Modifier le programme `trinome.py` en ajoutant les conditions nécessaires pour le calcul correct des racines dans le cas ou $\Delta > 0$. Penser à sauvegarder régulièrement.

Conclusion : En arithmétique flottante, il est souvent utile de réécrire les formules pour éviter d'importantes pertes de précision.

Attention au test d'égalité avec les flottants

Après avoir calculé sur votre feuille les descrimants des équations (2) et (3) essayer de les résoudre à l'aide de `trinome2.py`

$$x^2 + 0.4x + 0.04 = 0 \qquad (2)$$

$$x^2 - 0.13x + 0.004225 = 0 \qquad (3)$$

Vous l'aurez compris le test d'égalité `if delta == 0` ne fonctionne pas avec les flottants, pour s'en convaincre ajouter à la fin de votre programme la ligne `print delta`. Il est d'usage de réserver le test dégalité aux types scalaires discrets : *int, long, boolean*
Si x et y sont des flottants, pour tester l'égalité de x et y, on testera que la valeur absolue de $x - y$ est inférieur à une valeur choisie petite appelée epsilon notée $\mathcal{E}$.

1. Enregistrer `trinome2.py` sous `trinome3.py`
2. Modifier le code source de `trinome3.py` afin de remplacer le test d'égalité par l'utilisation de $\mathcal{E} = 10^{-8}$ et valider votre code avec l'équation (2).
3. Tous les problèmes ne sont pas encore réglés, loin de là, les équations suivantes possèdent chacune deux racines distinctes qu'elle est le problème si on essaye de les résoudre avec `trinome3.py`

$$0.1x^2 + 0.1999x + 0.0999 = 0 \qquad (4)$$

$$10^{-5}x^2 - 10^{-5}x - 10^{-5} = 0 \qquad (5)$$

Que faire quand le discriminant est litigieux ?

De manière générale tout trinôme à coefficients rationnels (comme ci-dessus) peut être transformé en un trinôme équivalent à coefficients "entiers" et dans ce cas la représentation flottante ne pose plus de problème, au moins pour le calcul du discriminant et si on s'assure que l'entier est exactement représentable dans le type choisi (ici flottant 64-bits cf TD_3)
Exemple (bien sûr à tester avec `trinome3.py`) :

- Pour l'équation (3) il est possible d'écrire : $10^6 x^2 - 1.3 \times 10^5 x + 4225 = 0$ et on obtient bien un descriminant égal à zéro.
- Le cas plus délicat des équations (4) et (5) est lui aussi résolu, bien que $\Delta < \varepsilon$ dans la forme de départ donc assimilé à une racine double, Δ devient différent de zéro une fois le trinôme réécrit levant ainsi l'ambiguité.

Si les coefficients sont *irrationnels* (font intervenir sqrt(2), sqrt(n), racines cubiques ou plus) on peut avoir des problèmes de ce genre et encore essayer de les corriger. Si les coefficients sont transcendants (pi, e, etc.) on ne sait pas faire grand-chose