Nombres réels, décimaux et flottants

Représentation des flottants sur des mots de taille fixe, mantisse, exposants

Tout nombre réel x non nul peut s'écrire sous la forme $s \times m \times 2^e$ où $s \in \{-, +\}$ est le signe de x, $m \in [1, 2[$ sa **mantisse**, en binaire, $e \in \mathbb{N}$ l'**exposant**.
On choisit ici de coder l'exposant sur 8 bits, en représentation décalée pour pouvoir représenter des nombres négatifs et positifs. Plus précisément, si l'exposant est x, on y ajoute 127 puis on code le nombre obtenu en binaire sur 8 bits ; si on veut lire un tel exposant en binaire, on en calcule la valeur en base 10, puis on retranche 127. Quel est le plus petit exposant représentable ? Le plus grand ?
Puisque m commence nécessairement par 1, on omet cette information et ne considère que les 23 chiffres suivants. Tout nombre représenté l'est ainsi sur 32 bits, dans l'ordre signe - exposant - mantisse.
Le nombre 0 n'est pas représenté dans ce système, c'est pourquoi des exceptions sont introduites. Le nombre 0 est représenté par 10...0 et par 0...0.
Sur 64 bits, la mantisse est représentée par 52 bits, l'exposant sur 11 bits.
exercice
Que représente 00110101101010111000011000010110 ? Et 10100111101010111001000011000011 ?
Comment représenter 145987 ? et 0,000006518 ?

Précision des calculs en flottants

On observe parfois des calculs erronés en Python.
Exemple 1

```
a=1/10
b = 3*a
```

Que vaut b ?

Les seuls nombres représentables de manière exacte sont ceux exprimables avec un nombre fini (et suffisamment petit) de chiffres en base 2, ce qui n'est pas le cas de 1/10. La valeur stockée dans la variable a n'est donc pas égale à 1/10, c'en est une approximation.
Quelle est la représentation de 0.1 sur 32 bits ?

Exemple 2 Calculer la somme des inverses des nombres de 1 à 10000, puis la même somme en partant des inverses des grands nombres vers les inverses des petits nombres. Obtient-on le même résultat ? Et si on calcule la somme jusque $n = 10000000$?

```
def sum(n):
    res = 0
    for i in range(1,n+1):
        res+=1/i
    return res

def sum2(n):
```

```
    res = 0
    for i in range(1,n+1):
        res+=1/(n+1-i)
    return res

n = 10000000
print(sum2(n)-sum(n))
```

D'où vient cet écart ?
Observons le code suivant :

```
n = -64
a = 1+ 2**n
b = a-1
c = 1+ 2**n -1

print(a,b,c)
```

Qu'oberve-t-on ?
Et si n est positif ?

Dans le second cas, Python a travaillé avec des grands entiers, et fournit donc un résultat correct.
Dans le premier, il doit travailler avec une représentation des nombres flottants.
Or pour $n = -64$, $2 * *n$ est très petit, et $1 + 2 * *n$ est trop proche de 1 pour en avoir une représentation distincte.

C'et ce qui permet d'expliquer également l'incongruité précédente : lorsqu'on ajoute les $\frac{1}{n}$, en partant de n petit, le dernier nombre que l'on ajoute est très petit par rapport à la somme déjà calculée, et risque de ne pas être pris en compte.

On veillera donc à :

- éviter d'additionner deux nombres d'ordres de grandeurs très différents (**absorption**)
- éviter les situations d'**élimination** : si on s'intéresse à la différence entre deux nombres de valeurs très proches, eux-même obtenus par calculs successifs, on peut obtenir un résultat manifestement faux.

  ```
  n = -64
  A = 1+2**n
  B = 1-2**n

  print(A,B,A-B)
  ```

  Qu'observe-t-on ?
- organiser le calcul des sommes pour ne pas avoir à ajouter à un nombre un autre beaucoup plus petit
- veiller à ce qu'un calcul, à défaut d'être exact, soit suffisamment proche du résultat attendu

Travaux pratiques

Calcul d'une approximation de π

Soit P_n le polygone régulier à n côté inscrit dans un cercle de centre O, de rayon 1. L'aire de ce polygone est n fois celle de chaque triangle de sommets deux points consécutifs du polygone et O.

1. L'aire d'un tel triangle vaut (en fonction de n) :
2. Vérifier que $\sin(\frac{x}{2}) = \sqrt{\frac{1-\sqrt{1-\sin^2(x)}}{2}}$
3. En déduire une relation de récurrence entre A_n et A_{2n}
4. Que vaut A_6 ?
5. Écrire une fonction qui, *en utilisant la relation de récurrence ci-dessus* calcule le $n$ème terme de la suite des aires des polygones à $2^n \times 3$ côtés inscrits dans le cercle de rayon 1. Tracer la courbe de ces aires en fonction de n. Qu'observez-vous ? Le résultat converge-t-il vers π comme attendu ? D'ou provient l'erreur ? Comment l'éviter ?

Éléments de correction

1. L'aire d'un triangle OBC isocèle en O, dont les côtés OB et OC sont de longueur 1 et dont l'angle en O est $\frac{2\pi}{n}$ est : $\frac{1\times(1\times\sin(\frac{2\pi}{n}))}{2} = \frac{\sin(\frac{2\pi}{n})}{2}$. La somme des angles en O de ces triangles vaut 2π et il y en a n : l'angle en O de chacun est donc $\frac{2\pi}{n}$.

2. La vérification de $\sin(\frac{x}{2}) = \sqrt{\frac{1-\sqrt{1-\sin^2(x)}}{2}}$ peut se faire de différentes manières. Tout d'abord, le contexte et la présence de la racine impose que le sinus observé soit positif, on prend donc x positif et "suffisamment petit".
   Ensuite, pour tout x, $\cos^2(x) = 1-\sin^2(x)$. Dans le cadre présent, $\cos(x) \geq 0$, donc $1-\sqrt{1-\sin^2(x)} = 1-\cos(x)$. Par les relations trigonométriques, $\cos(x) = \cos(\frac{x}{2}+\frac{x}{2}) = \cos^2(\frac{x}{2}) - \sin^2(\frac{x}{2}) = (1-\sin^2(\frac{x}{2})) - \sin^2(\frac{x}{2}) = 1-2\sin^2(\frac{x}{2})$, ce dont on déduit que $\sqrt{\frac{1-\sqrt{1-\sin^2(x)}}{2}} = \sqrt{\frac{1-\cos(x)}{2}} = \sqrt{\frac{2\sin^2(\frac{x}{2})}{2}} = \sin(\frac{x}{2})$.
3. Ici, l'énoncé manque de précision (toutes mes excuses !). On peut donc penser que A_n est l'aire d'un des triangles associés à P_n. Dans ce cas l'aire A'_n de P_n est donnée par $A'_n = nA_n$.
   D'après la question précédente, et puisque $A_n = \frac{\sin(\frac{2\pi}{n})}{2}$, on obtient $A_{2n} = \frac{\sin(\frac{2\pi}{2n})}{2} = \frac{\sqrt{\frac{1-\sqrt{1-\sin^2(\frac{2\pi}{n})}}{2}}}{2} = \frac{\sqrt{\frac{1-\sqrt{1-4A_n^2}}{2}}}{2}$
4. Pour calculer A_6, observons que chacun des six triangles que l'on peut former à partir de deux points consécutifs et de O est d'angle au sommet $\frac{\pi}{3}$. En conséquence, son aire vaut $A_6 = \frac{sin(\pi/3)}{2} = \frac{\sqrt{3}}{4}$.
5. La tentation est grande d'opter pour une fonction récursive, mais alors le tracer de la courbe serait peu efficace : on devrait calculer A_6, puis, pour calculer A_{12}, il faudrait encore calculer A_6, etc.
   On peut calculer à l'aide d'une boucle `for` les $A_{3\times 2^n}$ et les aires des polygones associés, puis ajouter ces derniers dans un tableau d'ordonnées, et renvoyer ce tableau.

```
def approxPi(m):
    aireTriangle = math.sqrt(3)/4
    aireT = 6*aireTriangle#aire dans le cas n = 1
    n = 6
    Aires = []
    Abs = []
    for i in range(1,m+1):
        aireTriangle = math.sqrt((1-math.sqrt(1-4*aireTriangle**2))/2)/2
        n = 2*n
        aireT = aireTriangle*n
        Aires.append(aireT)
        Abs.append(n)
    return Abs,Aires
```

```
#choix : tracer sans les abscisses Abs qui rendent illisible le graphe
def trace(n):
    abs,ord = approxPi(n)
    plt.plot(ord)
    plt.show()
```

Une valeur proche de π est bien trouvée, avant que l'algorithme ne fournisse des valeurs plus éloignées, pour finalement ne donner plus que des 0.
On a affaire ici à une situation d'élimination : lorsque l'aire A_n d'un petit triangle est très petite, la différence $1 - A_n$ est approximée en 1. L'aire A_{2n} vaut alors 0, et l'aire totale également.
Pour éviter cette situation, on peut utiliser la **quantité conjuguée**, c'est à dire observer que $\sqrt{1-\sqrt{a}} = \sqrt{1-\sqrt{a}\times\dfrac{1+\sqrt{a}}{1+\sqrt{a}}} = \sqrt{\dfrac{1-a^2}{1+\sqrt{a}}}$

Ici, $a^2 = 1 - \sin^2(2\pi/n)$, avec $\sin(2\pi/n) \geq 0$. Alors $A_{2n} = \dfrac{|A_n|}{\sqrt{2(1+\sqrt{1-4A_n^2})}}$

```
def approxPi2(m):
   aireTriangle = math.sqrt(3)/4
   aireT = 6*aireTriangle
   # print(aireTriangle,aireT)
   n = 6
   Aires = []
   Abs = []
   for i in range(1,m+1):
       aireTriangle = valabs(aireTriangle) /
math.sqrt(2+ 2*math.sqrt(1-4*aireTriangle**2))
       n = 2*n
       aireT = aireTriangle*n
       Aires.append(aireT)
       Abs.append(n)
   return Abs,Aires
```