Représentation des nombres

TP 10

Types de variables - v1.0

Lycée La Martinière Monplaisir, 41 Rue Antoine Lumière, 69372 Lyon

1 Conversion vers une base b quelconque

On peut décomposer n'importe quel entier en base b en une série de ses chiffres : par exemple $7924_{dec} = 7000_{dec} + 900_{dec} + 20_{dec} + 4_{dec} = 7_{dec} \times 10^3_{dec} + 9_{dec} \times 10^2_{dec} + 2_{dec} \times 10^1_{dec} + 4_{dec} \times 10^0_{dec}$.

Si on fait des divisions entières par la base 10 une première fois, on trouve 792 pour le quotient et 4 pour le reste, si l'on continue de travailler sur le quotient, on obtient 79 avec 2 comme reste, puis finalement 7 avec 9 comme reste et enfin 0 avec 7 comme reste : on se rend ainsi compte que les différents restes qui apparaissent à travers cet algorithme sont les chiffres dans la base 10, du moins significatif (celui de poids faible, i.e 4) au plus significatif (celui de poids fort, i.e 7).

Rappel : `22%5 -> 2` **et** `22//5 -> 4`

Question 1 Mettre en place une fonction `conversionBase10(n)` qui prend en entrée un entier positif `n` et renvoie une liste des chiffres selon l'algorithme décrit plus haut (donc dans l'ordre `[4, 2, 9, 7]` avec `n = 7924`).

Question 2 La tester rapidement avec 12, 57, 255, 7924.

Question 3 Mettre en place une fonction `conversionBaseb(n, b)` qui prend en entrée un entier positif `n` et une base `b` renvoie la liste des chiffres selon le même algorithme modifié.

Question 4 La tester rapidement avec 12, 57, 255, 7924 dans la base 2. Comparer avec ce qui est renvoyé par la commande `"{:b}".format(n)`.

Question 5 La tester rapidement avec 12, 57, 255, 7924 dans la base 16. Comparer avec ce qui est renvoyé par la commande `"{:X}".format(n)` pour l'hexadécimal.

Question 6 Que se passe-t-il quand on met 0 en entrée pour n ?

2 Somme d'entiers sur des formats limités

Commencer votre script par `import numpy as np`. Pour la suite on va travailler sur les entiers non signés `np.uint8` et `np.uint16`.

Question 1 Mettre en place une fonction `somme_ui8(L)` qui prend en entrée une liste d'entiers non signés sur 8 bits et renvoie leur somme sous forme d'un entier non signé sur 8 bits. On souhaite que la somme le long de la procédure soit bien un entier non signé sur 8 bits.

Remarque

Pour créer une liste d'entiers de type `np.uint8` à partir d'une liste d'entiers de type `int`, vous pouvez utiliser les commandes suivantes :

```
L0 = [12, 57, 255]
L1 = [np.uint8(x) for x in L0]
```

Question 2 Constater les problèmes discutés en cours et proposer des solutions.

Question 3 Mettre en place une fonction `somme_ui8_vers_ui16(L)` qui prend en entrée une liste d'entiers non signés sur 8 bits et renvoie leur somme sous forme d'un entier non signé sur 16 bits. On souhaite que la somme le long de la procédure soit bien un entier non signé sur 16 bits.

Question 4 Combien de 255 peuvent être sommés avec la fonction précédente sans atteindre le phénomène de dépassement d'entier ? Tester ces limites.

3 Conversion vers le format IEEE-754 simple précision

Vous trouverez dans votre espace de partage 2 fichiers à copier dans votre dossier de travail : `representation_binaire.py` et `conversion_etudiant.py`.

Question 1 Démarrer le script `conversion_etudiant.py` (raccourci `ctrl+shif+E`) et constater l'apparition dans la console d'une représentation binaire de π au format simple précision (bit de signe en rouge, bits d'exposant en bleu, bits de mantisse en vert).

Rappel :

- le signe doit être égal à 1 si $x < 0$ et 0 si $x > 0$;
- l'exposant doit être strictement positif si $|x| \geq 2$, strictement négatif si $|x| < 1$ et nul si on a déjà $1 \leq |x| < 2$;
- la mantisse doit être positive et doit être comprise entre 1 inclus et 2 exclu ;

Question 2 Écrire une fonction `nb2sem(x)` qui renvoie un triplet de nombres `s, e, m` correspondant au signe (0 ou 1), à l'exposant (un entier signé) et à la mantisse (un flottant) du nombre non-nul x.

Rappel :

- le décalage de l'exposant est de 127 pour qu'il soit toujours codé par un nombre positif sur 8 bits ;
- la mantisse possède une taille de 24 bits pour un flottant simple précision (23 bits mémoires et 1 implicite).

Question 3 En vous servant de la fonction `int2bin`, qui converti les entiers en chaine de caractères binaire, convertissez chacun des champs (s, e et m) en chaine de caractère binaire correspondant à la norme IEEE-754.

Question 4 Vérifiez la compatibilité avec la fonction `show_float("f", np.float32(x))` pour différentes valeurs de `x` (en particulier très grand et très petit).

4 Calcul d'un entier depuis ses chiffres et sa base

Question 1 Mettre en place une fonction `conversionDepuisBaseb(L, b)` qui prend en entrée une liste `L` de chiffres en base `b` et qui renvoie un entier n. Le premier élément de `L` sera le chiffre de poids faible (`[4, 2, 9, 7]` pour le nombre `7924`).

Question 2 Tester la fonction. Pour cela, vous pourrez utiliser les fonctions de l'exercice 1.

5 Énumération des listes binaires de taille n

Dans cet exercice, on recherche l'ensemble des listes binaires de taille n, une liste binaire étant définie comme une liste ne comprenant que des 1 et des 0.

`listes01(n)` doit renvoyer la liste de toutes les listes binaires de taille `n`. Par exemple, `listes01(3)` doit renvoyer `[[0, 0, 0], [1, 0, 0], [0, 1, 0], [1, 1, 0], [0, 0, 1], [1, 0, 1], [0, 1, 1], [1, 1, 1]]` (l'ordre est sans importance).

Question 1 Combien d'éléments doit contenir `listes01(n)` ?

Question 2 Mettre en place une fonction `listes01(n)` qui renvoie la liste de toutes les listes binaires de taille `n`. Vous pourrez vous aider de `conversionBaseb(n, b)` avec `b=2`.

Question 3 Tester la fonction.

6 Approximation de la dérivée d'une fonction hors du point zéro

D'après la formule de Taylor, on a la relation suivante, pour h suffisamment petit [1] :

$$f(x+x\cdot h) \approx f(x) + f'(x)\,(x\cdot h) + f^{(2)}(x)\frac{(x\cdot h)^2}{2} + f^{(3)}(x)\frac{(x\cdot h)^3}{6}$$

On a donc l'approximation de la dérivée d'une fonction [2] :

$$f'_{approx}(x) = \frac{f(x+x\cdot h) - f(x-x\cdot h)}{2\cdot x\cdot h} \approx f'(x) + 2\cdot f^{(3)}(x)\frac{(x\cdot h)^2}{6}$$

Question 1 Écrire la fonction `derivee(f,x,h)` qui permet le calcul de la dérivée au point `x` de la fonction `f`.

Il y a 2 principales sources d'erreurs, toutes les 2 dépendantes de h :

- l'erreur prévue par la formule de Taylor, d'environ $f^{(3)}(x)\frac{(x\cdot h)^2}{3}$;
- l'erreur de précision due à la soustraction des flottants, d'environs $\frac{f(x)\varepsilon}{2\cdot x\cdot h}$ (avec ε l'erreur relative d'un calcul avec des flottants).

1. Les conditions sur f, la signification précise du terme « suffisament petit » ainsi que de la signification précise de la notation $\approx$ dans ce contexte est à mettre en relation avec vos cours de mathématiques, passés ou futurs.
2. inspiré de la partie III.B.1 du sujet de la banque Centrale 2019

Question 2 Sachant que les nombres flottants en Python sont classiquement calculés avec 53 bits de mantisse (52 stockés et 1 bit implicite), quelle est la valeur maximale de ε ? Aide : le plus grand écart relatif qu'il existe entre 2 flottants consécutifs se retrouve, entre autre, entre 1 et le nombre directement plus grand que 1.

Question 3 Vérifier votre résultat en important le module `sys` et en vérifiant ce que renvoie la commande `sys.float_info.epsilon`.

Une analyse présentée en annexe permet de déduire qu'une valeur pertinente de h semble être $\sqrt[3]{\varepsilon}$.

Pour tester notre approximation, nous allons l'utiliser avec une fonction dont on connait la valeur de la dérivée : $f(x) = \exp(-x)$.

Pour la suite, on prendra des valeurs de h allant de 10^{-15} à 10^{0}. On créera cet ensemble de valeur grâce à la commande `np.logspace(-15, 0, 8)` qui renvoie une liste de 8 valeurs équitablement réparties sur une échelle logarithmique.

Question 4 Tester la fonction `derivee` pour la fonction $f(x) = \exp(-x)$ pour le point $x = 1$ pour les différentes valeurs de h.

Question 5 Écrire une fonction `rech_hopt(x, Lh)` qui renvoie la valeur de h qui a minimisé l'erreur pour le calcul de la dérivée au point x de la fonction $f(x) = \exp(-x)$. Est-on très éloigné de la valeur $\sqrt[3]{\varepsilon}$? Vérifier pour plusieurs valeurs de x.

7 π vraiment irrationnel ?

La fraction continue de π peut être utilisée pour générer les meilleures approximations rationnelles successives :

$$\frac{3}{1}, \frac{22}{7}, \frac{333}{106}, \frac{355}{113}, \frac{103993}{33102}, \frac{104348}{33215}, \frac{208341}{66317}, \frac{312689}{99532}, \frac{833719}{265381}, \frac{1146408}{364913}, \frac{4272943}{1360120}, \frac{5419351}{1725033}, \ldots$$

Vous trouverez dans votre espace de partage 2 fichiers : `approximation_rationelle_pi.txt` et `pi_etudiant.py`. Copiez-les dans votre espace de travail et démarrez le script Python (raccourci clavier : `ctrl+shift+E`). Par la lecture du fichier `approximation_rationelle_pi.txt`, 2 listes `num` et `den` sont créées qui correspondent aux approximations rationnelles successives. Ainsi $\forall i \in [\![0, len(L) - 2]\!]$, la fraction $\dfrac{num[i+1]}{den[i+1]}$ est plus proche de π que $\dfrac{num[i]}{den[i]}$.

Question 1 En utilisant les fonctions `np.float16` et `np.float32` sur π (accessible avec `np.pi`), constater que le nombre de chiffres affichés n'est pas du tout le même.

Question 2 Écrire une fonction `approx_ratio_lim(conversion)` qui renvoie le premier couple de valeurs du numérateur et du dénominateur qui correspond à une égalité entre la division convertie et la valeur de π convertie. Cette fonction prend en entrée la fonction de conversion (`np.floatxx`).

Question 3 Constatez que la valeur du dénominateur est beaucoup plus faible quand l'approximation est faite avec un flottant demi-précision (`np.float16`) qu'avec un flottant simple précision (`np.float32`), lui-même plus faible que pour un flottant double-précision (`np.float64`).

En réalité, l'approximation de π stockée en mémoire est de la forme $\dfrac{N}{2^k}$.

Question 4 Écrire une fonction `approx_rat(x)` qui renvoie le couple de valeur (N, k) correspondant au flottant x. Pour cela, on répétera des multiplications par 2 jusqu'à ce que la partie fractionnaire soit nulle.

Rappel : la mantisse a une taille de 11 bits pour un flottant demi-précision, 24 bits pour un flottant simple précision et 53 bits pour un flottant double précision.

Question 5 En testant cette fonction, constater la cohérence des valeurs de k avec le nombre de bits significatifs de la mantisse.

8 Conversion vers une base en récursif

Question 1 Réécrire la fonction `conversionBaseb(n, b)` dans une version récursive.

9 Problème sur le panier

On souhaite vérifier que le prix d'un panier de produits correspond bien à la valeur affichée [3].

On considérera ici que le prix du panier correspond à des articles dont les prix sont stockés dans une liste `L` (par exemple, `L = [4.99, 15.99, 21.42]`).

Question 1 Calculer le prix du panier correspondant en le mettant dans une variable `S`.

Pour l'affichage, on utilise une chaine de caractères avec 2 chiffres après la virgule : la commande permettant la création de cette chaine est `":0.2f".format(S)`.

Question 2 Convertir votre somme en chaine de caractère, puis la reconvertir en nombre : ce nombre est-il le même que `S` ?

Question 3 Proposer des solutions et vérifier leurs mises en œuvre.

10 Mesure du temps avec un flottant

Le module `numpy` est à importer au début du script en le renommant `np`.

On souhaite utiliser une variable évoluant en fonction du temps passé [4]. Cette variable sera un flottant simple précision (`np.float32`) : toutes les millisecondes, cette variable sera incrémentée de `np.float32(0.001)`.

Question 1 Au bout de combien de temps a-t-on un décalage de 1 `ms` ? Pour répondre, il est nécessaire d'avoir une variable entière qui compte le nombre de millisecondes en parallèle de la variable flottante présentée précédemment.

Question 2 Proposer une modification de votre algorithme précédent permettant de stocker la différence entre valeur réelle et valeur attendue. Est-ce que la dérive est constante ?

Question 3 Vérifier que ce décalage arrive bien plus vite avec des formats de flottants plus petits demi-précision `np.float16`.

3. Inspiré de https://blog.pascal-martin.fr/post/deux-nombres-flottants-egaux-ne-sont-pas-egaux/
4. Inspiré de https://www-users.cse.umn.edu/~arnold/disasters/Patriot-dharan-skeel-siam.pdf

Annexe

Si l'on souhaite une erreur totale minimum, il faut atteindre un minimum pour la fonction $e(h)$:

$$e(h) = |f^{(3)}(x)|\frac{(x \cdot h)^2}{3} + \frac{|f(x)|\varepsilon}{2 \cdot |x| \cdot h}$$

En dérivant pour analyser cette fonction, on obtient :

$$e'(h) = \frac{2}{3} \cdot x^2 |f^{(3)}(x)| \cdot h - \frac{1}{2} \cdot \left|\frac{f(x)}{x}\right| \cdot \frac{1}{h^2} \cdot \varepsilon$$

L'annulation de la dérivée, correspondant à un minimum de l'erreur $e(h)$, nous donne :

$$e'(h_{opt}) = 0 \quad \text{pour} \quad h_{opt}^3 = \frac{3}{4} \cdot \left|\frac{f(x)}{x^3 f^{(3)}(x)}\right| \cdot \varepsilon \quad \text{donc} \quad h_{opt} = \sqrt[3]{\frac{3}{4} \cdot \left|\frac{f(x)}{x^3 f^{(3)}(x)}\right| \cdot \varepsilon}$$

On va faire l'hypothèse que les fonctions que l'on souhaite étudier possèdent un effet d'échelle raisonnable, similaire aux fonctions trigonométriques et exponentielles telles que $A_0 \cos(\omega t)$ ou $A_0 \exp\left(-\frac{t}{\tau}\right)$. On peut ainsi définir une grandeur $\Delta(f) > 0$ tel que :

- les valeurs de x qui vont nous intéresser sont proches de $\Delta(f)$ (c.-à-d. $\frac{|x|}{\Delta(f)}$ proche de l'unité dans l'intervalle d'étude) ;
- $|f^{(k)}|$ est proche de $\Delta(f)^{-k} \times |f|$ dans l'intervalle d'étude.

On a, pour les exemples cités plus tôt : $\Delta\left(A_0 \cos(\omega t)\right) = \frac{1}{\omega}$ et $\Delta\left(A_0 \exp\left(-\frac{t}{\tau}\right)\right) = \tau$.

On a donc $\left|\frac{f(x)}{x^3 f^{(3)}(x)}\right| \approx 1$. Et on peut déduire qu'une valeur pertinente de h *a priori* est donc :

$$h_{opt} = \sqrt[3]{\varepsilon}$$