TP : Représentation des nombres flottants

L'objectif de ce problème est de représenter graphiquement les nombres flottants codés sur 8 bits.

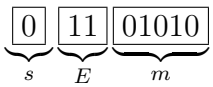
1 Le codage des flottants sur 8 bits

Nous supposons ici que les nombres flottants sont codés sur 8 bits de la façon suivante :

$$x = (-1)^s \times 1.(m_1 m_2 \dots m_5)_2 \times 2^e.$$

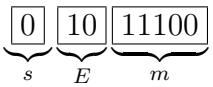
- l'entier s appelé **signe** vaut 0 si x est positif et 1 sinon.
- les cinq bits $m_1, \dots, m_5$ constituent la **mantisse** m.
- l'entier e est l'**exposant**, on le code par la représentation binaire sur 2 bits de l'entier *naturel* E défini par $E = e + 1$. L'entier E étant codé sur 2 bits, il prend les valeurs de $[\![0, 2^2 - 1]\!] = [\![0, 3]\!]$ et donc $e \in [\![-1, 2]\!]$.

Par exemple, le nombre décimal 5.25 s'écrit en base deux 1.0101×2^2. On a donc $s = 0$, $m = 0101$ et $e = 2$ donc $E = 3$ donc sa représentation sur 8 bits est :



Remarque : on ne pourra pas représenter le nombre 0.

1. Quel est le nombre dont la représentation sur 8 bits est :



2. Donner la représentation flottante sur 8 bits du nombre $\frac{29}{8}$.

2 Quelques fonctions auxiliaires

3. Si `m` est une chaîne de caractères, et i et j deux entiers avec $i < j$, alors l'instruction `m[i:j]` renvoie une tranche de `m`, la séquence composée des caractères `m[k]` pour $i \leqslant k < j$. Par exemple, si `m='superPython'`, `m[2:7]` renvoie la chaîne de caractères `'perPy'`. On parle de «slicing» (trancher en Anglais).

   Compléter la fonction `tranche` suivante qui prend en argument `m` une chaîne de caractères, i et j deux entiers avec $i < j$ et qui renvoie le même résultat que `m[i:j]` mais *sans utiliser cette instruction*.

   Par exemple, `tranche('superPython', 2, 7)` renverra `'perPy'`.

```
def tranche(m, a, b):
    sous_mot = '' # chaîne de caractère vide
    for ??????
        sous_mot = ??????
    return ??????
```

Dans toute la suite du problème, on pourra si besoin utiliser **librement** le «slicing» ou la fonction `tranche`.

4. Compléter la fonction suivante `VersBaseDeux` qui prend en argument n un entier naturel et renvoie la chaîne de caractère correspondant à son écriture binaire. Par exemple, 13 vaut $(1101)_2$, ainsi `VersBaseDeux(13)` renverra la chaîne `'1101'`.

```
def VersBaseDeux(n):
    """Données: n un entier naturel n
       Résultat: une chaîne de caractère représentant l'écriture en base 2 de n """
    if n == 0:
        return '0'
    mot = ''
    while n  ????
        bit = ????
        mot = ????
        ??? = n // 2
    return mot
```

5. En déduire une fonction `DecimalVers8bits` qui prend en argument un entier n compris entre 0 et 255 et qui renvoie le mot binaire de 8 bits qui le représente. Par exemple, le nombre 13 a pour écriture binaire $(1101)_2$ et donc `DecimalVers8bits(13)` renverra le mot `'0000 1101'` (on sera attentif au fait que l'on doit forcément renvoyer un mot de 8 bits quitte à rajouter des zéros sur la gauche).

6. Écrire une fonction `VersBaseDix` qui prend en argument m un mot binaire (une chaîne de caractères constituée de 0 et de 1) et renvoie l'entier naturel codé par ce mot binaire. Par exemple, si m vaut `'1101'`, `VersBaseDix(m)` renverra 13.

   Si besoin et on pourra l'utiliser librement, on rappelle que si `a` est un chiffre l'instruction `int('a')` convertit la chaîne de caractères `'a'` en l'entier `a`.

3 Représentation graphique

Avec 8 bits, on peut coder tous les entiers naturels de 0 à 255. Par exemple, l'entier 106 est représenté par le mot binaire de 8 bits `'01101010'`. Nous avons vu à la première section que ce mot binaire représente aussi le nombre flottant $5,25$. En parcourant, tous les entiers de 0 à 255, on va donc pouvoir fabriquer tous les mots binaires de 8 bits et donc tous les nombres flottants sur 8 bits.

A l'aide du module `matplotlib.pyplot` nous allons représenter graphiquement les 256 nombres flottants codés sur 8 bits.

7. Écrire une fonction `signe` qui prend en argument un mot binaire de 8 bits `m` et qui renvoie l'entier 1 si le premier bit de `m` (celui de gauche) vaut `'0'` et qui renvoie l'entier -1 si le premier bit vaut `'1'`. Par exemple, `signe('01101011')` renverra 1.

8. Écrire une fonction `exposant` qui prend en argument un mot binaire de 8 bits `m` et qui renvoie l'exposant e de l'écriture scientifique du nombre flottant représenté par le mot `m`. Par exemple, le mot binaire `'01101010'` représente le nombre flottant 5.25 dont l'écriture scientifique binaire est $(1.0101)_2 \times 2^2$. L'exposant vaut donc 2, ainsi `exposant('01101010')` renverra l'entier 2.

9. Écrire une fonction `mantisse` qui prend en argument un mot binaire de 8 bits et qui renvoie le nombre flottant f dont l'écriture binaire est $(0.m_1m_2m_3m_4m_5)_2$ où m_1, m_2, m_3, m_4 et m_5 sont les cinq derniers bits de `m`. Par exemple `mantisse('01101010')` renverra le nombre flottant f dont l'écriture binaire est $(0.01010)_2$, c'est-à-dire $f = \frac{1}{4} + \frac{1}{16} = 0.3125$.

10. En déduire une fonction `EntierVersFlottant` qui prend en argument un entier n compris entre 0 et 255 et qui renvoie le nombre flottant représenté par le mot binaire de 8 bits codant n. Par exemple `EntierVersFlottant(106)` renverra le nombre flottant 5.25.

11. Compléter ainsi le script suivant qui calcule la liste X des 256 nombres flottants représentés par tous les mots binaires de 8 bits, puis représente tous les nombres flottants sur l'axe des abscisses. On obtient alors la figure ci-dessous.

```
import matplotlib.pyplot as plt

X = ?????
Y = ?????

plt.??????(????, ????, 'r.')
plt.show()
```

4 Pour les plus rapides

On se propose ici d'écrire de réécrire en récursif les fonctions 4 et 6 déjà vues précédemment

12. Écrire une fonction récursive `VersBaseDeuxRec` qui prend en argument n un entier naturel et renvoie la chaîne de caractère correspondant à son écriture binaire.

13. Écrire une fonction `VersBaseDixRec` qui prend en argument m un mot binaire (une chaîne de caractères constituée de 0 et de 1) et renvoie l'entier naturel codé par ce mot binaire.

14. Déterminer la complexité des fonctions 4 et 6.

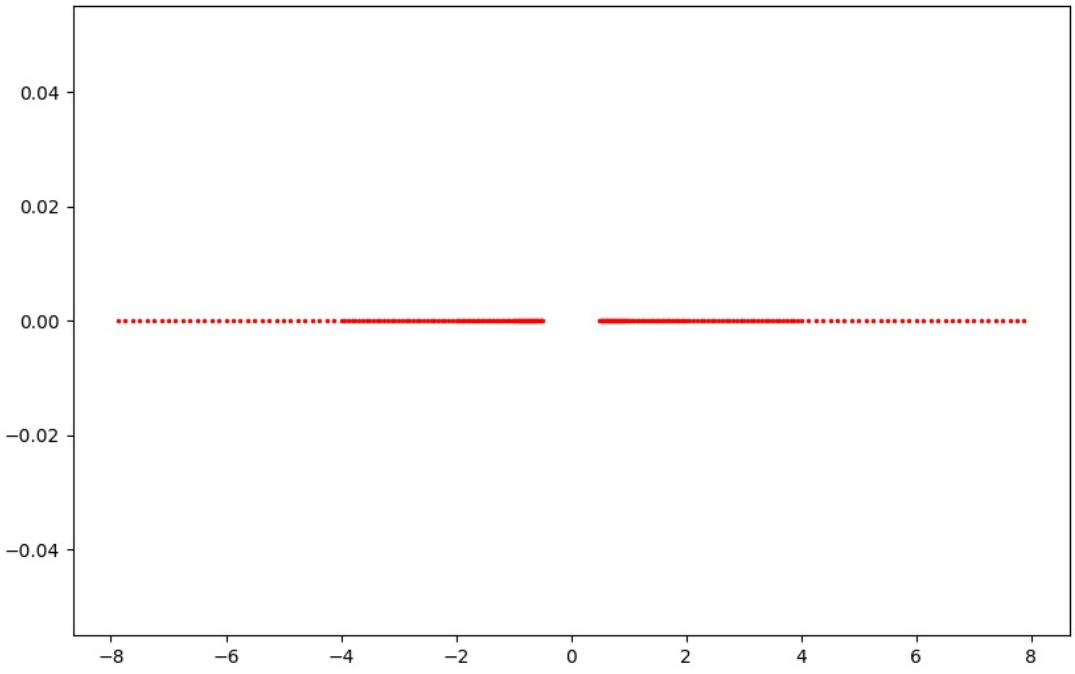


FIGURE 1 – Nombres flottants sur 8 bits