Année 2023 - 2024 SHPI 2

Méthodes numériques appliquées

TP 1

Voir `Moodle` pour les consignes techniques, et pour le dépôt en fin de séance. Sauf problème technique avéré, vous devez travailler avec `Jupyter` et donc déposer au format `ipynb`.

Vous aurez besoin, quelque part dans cet énoncé, du résultat suivant (admis ici) : pour tout réel x, la série numérique $\sum_{n\geqslant 0} \frac{x^n}{n!}$ converge, et sa somme vaut e^x.

L'objectif n'est pas de finir à tout prix, mais de traiter correctement les problèmes posés.

Attention à soigner la présentation de votre carnet, en utilisant les exemples au format `markdown` *du fichier* `Modele.ipynb` *disponible dans* `Moodle`*. Et en particulier s'il s'agit d'écrire des mathématiques.*

1 Représenter les flottants

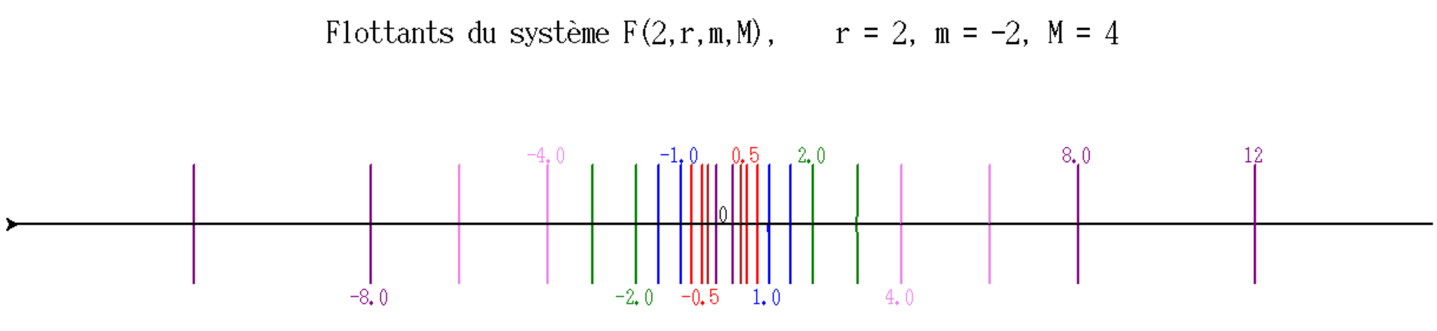
Les nombres à virgule flottante (ou "flottants") ont été vus en cours/TD : si l'on choisit la base $b = 2$, ce sont les nombres du système $\mathcal{F} = \mathcal{F}(2, r, m, M)$, c'est-à-dire ceux de la forme

$$\pm\,(0.1\,d_{-2}\cdots d_{-r})_2 \times 2^j, \quad \text{avec } m \leqslant j \leqslant M, \text{ et pour } \ k > 1,\ d_{-k} = 0 \ \text{ ou } 1$$

auxquels on ajoute 0. Les "flottants-jouets" sont des cas particuliers, où l'on prend de petites valeurs de r, m et M. C'est ce qu'on va faire ici, pour pouvoir représenter *tous* les flottants du système choisi, sur un axe gradué.

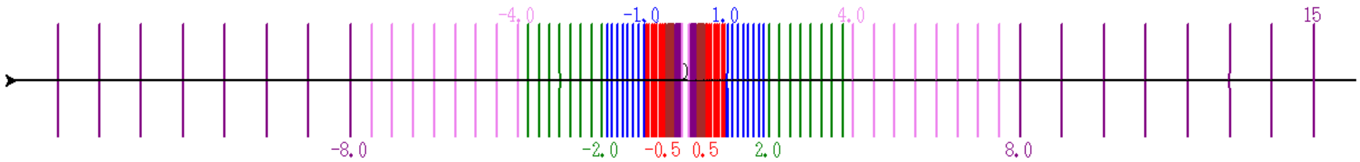
L'outil utilisé sera la librairie `turtle` de Python : elle permet de faire bouger dans le plan de l'écran une tortue virtuelle ; si le crayon de la tortue est baissé, chaque déplacement crée un dessin. Les unités par défaut sont le pixel pour les longueurs, le degré pour les angles ; voir https://docs.python.org/3.9/library/turtle.html

Votre objectif sera de faire réaliser à la tortue un dessin du type ci-dessous :

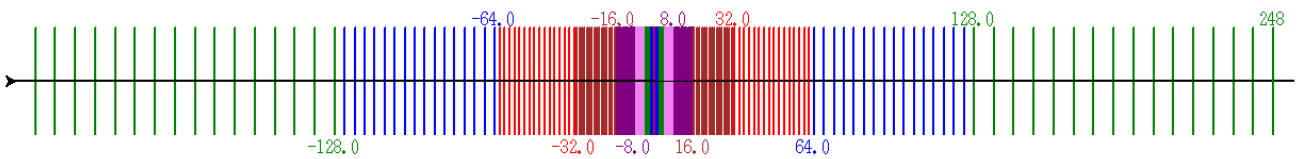


donc avec un trait vertical à l'endroit de chaque flottant du système (c'est l'objectif prioritaire) et autant que possible : des couleurs différentes pour les différentes valeurs de l'exposant j, l'indication de la valeur numérique de certains de ces flottants, et en particulier celle du plus grand. Les entiers r, m et M devront être en paramètres, de façon à pouvoir obtenir avec la *même* fonction d'autres cas, par exemple :

Flottants du système F(2,r,m,M),   r = 4, m = -3, M = 4



Flottants du système F(2,r,m,M),   r = 5, m = -4, M = 8



N. B. La première fois que vous lancerez un dessin avec la tortue, `Jupyter` ouvrira une fenêtre graphique séparée. *Laissez cette fenêtre ouverte* mais insérez au début de chaque programme de dessin la commande `reset()` qui efface le dessin précédent et ramène la tortue au centre.

Vous aurez besoin d'abord de répondre aux questions suivantes (dans une cellule `markdown` de votre compte rendu) : avec les notations ci-dessus, pour le système $\mathcal{F}(2, r, m, M)$,

a) quel est le plus petit flottant strictement positif ?

b) quel est le plus grand flottant ?

c) si on fixe la valeur de l'exposant j et qu'on ne regarde que ces flottants-là, que peut-on dire de l'écart entre deux flottants consécutifs ?

d) pour les flottants strictement positifs, quel écart y a-t-il entre le plus grand flottant d'exposant j et le plus petit flottant d'exposant $j+1$?

2 Une série à problèmes

On considère l'algorithme suivant :

$s := 1\,;\ t := 1\,;\ n := 1$

Tant que $|t| > \varepsilon$ faire $\{\ t := t \times \dfrac{x}{n}\,,\ s := s + t\,,\ n := n + 1\ \}$

Afficher s

Que fait cet algorithme ? Programmez-le, testez-le pour différentes valeurs de x (les différents tests devront figurer dans le compte rendu). Que donne-t-il par exemple pour $x = -40$, $\varepsilon = 10^{-12}$? Pour quelles valeurs de x donne-t-il le résultat attendu[1], avec une erreur relative inférieure à 10^{-3} ? Expliquez[2] ce qui se passe quand l'erreur relative devient trop grande.

[1]Dans votre compte rendu, on devra savoir d'où vous tirez ce « résultat attendu » (la bonne valeur numérique) : votre calculatrice ? (mais est-elle plus fiable que l'algorithme ?) un site internet ? Python lui-même ? Cette dernière solution serait préférable.

[2]Attention, il n'est pas écrit « décrivez »...

3 Arithmétique

1. On donne la fonction suivante en `Python` :

```
def fonction(nb : int)-> bool :
    resultat = False
    if nb != 1 :
        resultat = True ; cpt = 2 ; r = sqrt(nb)
        while resultat and cpt <= r :
            if nb % cpt == 0 : resultat = False
            cpt += 1
    return resultat
```

   Que fait cette fonction ? Vérifiez-le sur quelques exemples bien choisis, que vous laisserez dans votre carnet. Pour la condition `cpt <= r` : pourquoi peut-on la mettre, sans changer le résultat global ? Et pourquoi est-ce intéressant de la mettre ?

2. On pose $N = 36\,028\,797\,018\,963\,971$. Que retourne `fonction(N)` ? *Théoriquement*, combien d'opérations élémentaires ont été nécessaires ? on ne comptera pas les affectations, et le calcul de la racine carrée sera, pour simplifier, compté comme une seule opération.

   Combien d'opérations seraient théoriquement nécessaires pour calculer `fonction(N**2)` ?

3. Dans `Jupyter`, une méthode possible pour mesurer un temps de calcul est de taper dans une cellule de code `%timeit` suivi de l'instruction qu'on veut chronométrer. Mesurez ainsi le temps de calcul de `fonction(N)`, et donnez une estimation du temps qu'il faudrait pour calculer `fonction(N**2)`.

   Vous *admettrez* que le résultat de ce dernier calcul est `True`. Est-ce normal ? Pouvez-vous expliquer ce qui se passe ? Comment modifier l'algorithme pour corriger ce problème ?