Exercice 1 : La factorielle, une fonction récursive classique

Il existe plusieurs façons de calculer la factorielle d'un entier. Dans cet exercice, on s'intéresse à une méthode « sans boucle ». Pour cela, on s'appuie sur une propriété bien connue de la factorielle, à savoir :

$$\forall n \in \mathbb{N}*, \quad n! = n \times (n-1)!.$$

L'idée est donc de déplacer le problème : pour calculer la factorielle d'un entier n, on prend cet entier et on le multiplie par le résultat de la factorielle de $n-1$. Celle-ci sera calculée de la même manière : il s'agira du produit de $n-1$ avec la factorielle de $n-2$. On continue ainsi jusqu'à tomber sur la factorielle de 1, dont le résultat est immédiat.

1. Écrire une fonction Scilab `factorielle(n)` qui admet en entrée un entier n et qui renvoie 1 si n vaut 1, et n sinon.
2. Modifier la fonction précédente pour qu'elle renvoie ce qu'on veut, à savoir la factorielle de n. *Indication : On peut faire appel à une fonction dans sa définition elle-même. Par exemple, la fonction `factorielle()` peut être appelée à l'intérieur de la fonction `factorielle()` elle-même.*

Exercice 2 : Ensemble de Cantor

Pour construire l'ensemble de Cantor, on part de l'intervalle $[0, 1]$. La première itération consiste à découper cet intervalle en deux sous-intervalles : $[0, 1/3] \cup [2/3, 1]$. En fait, on coupe l'intervalle en trois parties égales, et on retire la partie centrale. À chaque itération, on renouvelle ce procédé : on découpe tous les sous-intervalles pour retirer leurs parties centrales.

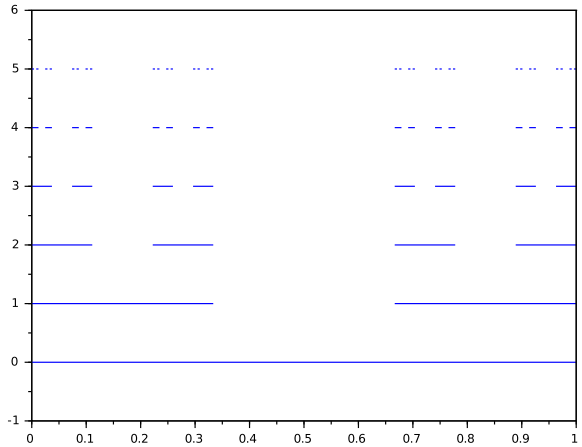

FIGURE 1 – Les 5 premières itérations de l'ensemble de Cantor.

L'ensemble de Cantor est la limite de ce procédé : on le réitère une infinité de fois. Dans la pratique, nous allons bien évidemment l'arrêter au bout d'un certain nombre d'itérations. Comme pour l'Exercice 1, nous allons dans un premier temps écrire une fonction qui ne fait pas tout à fait ce qu'on veut avant de la modifier.

1. Écrire une fonction Scilab `cantor(segment, iter, itermax)` qui admet trois arguments en entrée :

— `segment`, un vecteur colonne à deux éléments (les abscisses du segment à découper) ;

— `iter`, l'itération actuelle de la construction ;

— `itermax`, le nombre d'itérations à effectuer.

Si `iter` est égal à `itermax`, alors on a atteint la dernière itération : on renvoie le segment lui-même. Dans le cas contraire, on découpe le segment en trois parts égales, et on renvoie les deux parties extrémales (sous la forme d'une matrice dans laquelle chaque colonne représente les abscisses des extrémités d'un sous-intervalle).

2. Modifier la fonction précédente : au lieu de renvoyer les deux sous-intervalles, on renverra plutôt le résultat de leurs découpages par la fonction `cantor()` elle-même. On pensera bien à incrémenter la valeur de la variable `iter`.

3. Écrire un script Scilab permettant d'afficher une itération donnée de l'ensemble de Cantor. La fonction `cantor()` renvoie les abscisses des segments à afficher : on pourra par exemple associer des ordonnées à ces abscisses et utiliser la fonction `plot()`.

Exercice 3 : Flocon de Koch

La construction du flocon de Koch est similaire à celle de l'ensemble de Cantor, dans le sens où il s'agit, à chaque itération, de remplacer un segment par d'autres. Un segment donné est toujours divisé en trois parties égales. La partie du milieu est remplacée par un triangle équilatéral (dont on a retiré la base, voir Figure 2).

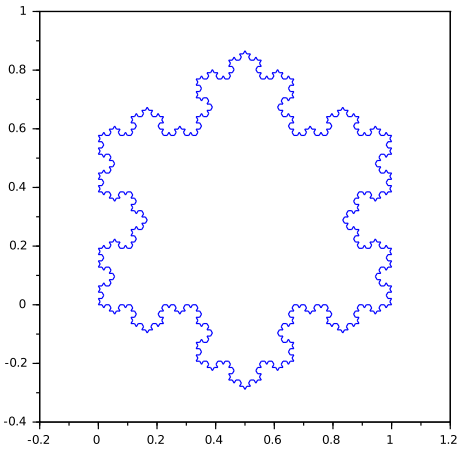

FIGURE 2 – Flocon de Koch, après 4 itérations.

1. Déterminer (sur le papier) les coordonnées du sommet manquant du triangle équilatéral à construire, en fonction des coordonnées des extrémités du segment.

2. Comme dans l'Exercice 2, écrire une fonction récursive pour construire le flocon de Koch.