Nage du Requin

Explication du code

* Importation des modules utiles
  + *math* 
    - Elle nous permet d’avoir accès à un catalogue de fonction intéressantes et fonctionnelles à notre problème tels que math.log pour le logarithme népérien ou des fonctions math.cos ou sin pour le cosinus ou sinus.
  + *random*
    - Ce package nous donne accès à pleins de fonction intéressantes, qui permet la génération de nombres pseudo-aléatoire. Dans ce module, nous nous intéresserons à deux fonctions
    - *choice*
      * Permet de choisir un élément au hasard dans une liste.
      * Paramètre
        + Une liste

Une liste

* + - * Renvoie
        + Un élément de la liste

Peut être un nombre réel, une liste, une chaine de caractère, un dictionnaire, un objet…

* + - *uniform*
      * Elle renvoie un nombre réel au hasard dans un intervalle donné.
      * Paramètre
        + Borne inférieure de l’intervalle

Un nombre réel

* + - * + Borne supérieur de l’intervalle

Un nombre réel

* + - * Renvoie
        + Un nombre aléatoire entre la borne inf et sup de l’intervalle

Un nombre réel

* + *matplotlib*
    - Pour pouvoir faire des représentations graphiques
    - *pyplot*
      * + Pour tracer un graphe avec des valeurs x et des valeurs y en reliant les points.
  + *scipy*
    - Utilisation pour le calcul scientifique.
    - *stats*
      * Contient un grand nombre de distributions à la fois continues et discrètes
* *class Requin*
  + Création et manipulation du requin
  + d*ef \_\_init\_\_* prend comme paramètre **iteration**, **p**, **alpha**, **sigma** et ne renvoie rien.
    - Initialisation des variables entrés par l’utilisateur, avec vérification de celle-ci
    - Ces paramètres sont entrés par l’utilisateur, donc une vérification sur les paramètre est impératif, on ne veut pas que l’utilisateur ne rentre n’importe quoi. C’est ce que nous avons fait dans une autre fonction appelé *vérification*
      * Paramètre
        + **iteration** : Le nombre d'itérations désirées

Doit être un entier naturel

* + - * + **p** : la probabilité p issu de la loi de Bernouilli, qui est la mémoire du requin

Doit être un nombre réel

* + - * + **alpha** : le coefficient de la loi symétrique alpha stable

Doit être un nombre réel

* + - * + **sigma** : permet de vérifier que c’est une loi alpha stable et est égale à 1 par défaut.

Doit être un nombre réel

* + *def which\_case* prend aucun paramètre mais renvoie une chaîne de caractère.
    - On cherche dans quel cas on se trouve. Pour trouver dans quel cas on est, il faut seulement multiplier alpha et p, et si le résultat est inférieur à 1 alors on se trouve dans un cas sous critique, ou si le résultat est de 1 alors on est dans le cas critique et si le résultat est supérieur à 1 alors on est dans le cas super critique
      * Retourne
        + Renvoie le cas dans lequel on est

Doit être une chaine de caractère

* + *def reset* ne prend pas de paramètre et ne renvoie rien.
    - On réinitialise les variables. On vide nos requin (requin + convergence)
  + *def verification* ne prend pas de paramètre et ne renvoie rien.
    - On vérifie les variables qui ont été manipulés par l’utilisateur, on n’accepte pas n’importe quel valeur pour chaque variable.
    - La mémoire doit être entre 0 et 1.
    - Le nombre d’itération doit être dans
    - L’alpha doit être entre strictement 0 et 1.
    - Sigma doit être positif, il n’y a pas vraiment de condition sur sigma, on le laissera à 1.
  + *def modifier* prend comme paramètre**iteration**, **p**, **alpha**, **sigma** et ne renvoie rien.
    - On peut modifier les variables entré plutôt par l’utilisateur pour éventuellement faire différentes simulations en modifiant la variable concerné. Bien sûr, après que l’utilisateur a modifié les variables, une vérification sur celles ci est nécessaire.
      * Paramètre
        + **iteration** : Le nombre d'itérations désirées

Doit être un entier naturel

* + - * + **p** : la probabilité p issu de la loi de Bernouilli

Doit être un nombre réel

* + - * + **alpha** : le coefficient de la loi symétrique alpha stable

Doit être un nombre réel

* + - * + **sigma** : permet de vérifier que c’est une loi alpha stable et est égale à 1 par défaut.

Doit être un nombre réel

* + *def build* ne prend aucun paramètre et ne renvoie rien.
    - On construit le requin ainsi que sa convergence, en appelant la fonction *buildRequinAndConvergence*, qui nous renvoie deux éléments.
  + *def calcul\_X* prend comme paramètre **U** et **V** et renvoie un nombre réel.
    - Calcule l’expression de X qui est la suivante (écrire la formule de X)
      * Retourne
        + Calcule la valeur de X

Doit être un nombre réel

* + *def convergence\_cases* prend comme paramètre **S\_n\_i** et **i** et renvoie un nombre réel.
    - Calcule le comportement asymptomatique du requin au temps i en fonction du cas (sous critique, critique ou super critique).
      * Paramètre
        + **S\_n\_i** : La valeur du requin au ième temps

Doit être un nombre réel

* + - * + **i** : Le ième temps du requin

Doit être un entier

* + - * Retourne
        + La valeur de la convergence du requin au temps i

Un nombre réel

* + *def buildRequinAndConvergence* ne prend pas de paramètre et renvoie deux variables qui sont des listes.
    - Création du requin et de sa convergence en utilisant à la fois les fonction crées précédemment et des paramètres entrés par l’utilisateur.
    - *def initialisation* prend deux paramètres : **U** et **V**et renvoie un nombre réel.
      * On définit le point de départ du requin, on dira qu’il commence à 0 et avancera au premier temps vers la valeur X
      * Paramètre
        + **U** : Une uniforme sur -1; 1

Doit être un nombre réel

* + - * + **V** : Une autre uniforme sur 0; 1

Doit être un nombre réel

* + - * Retourne
        + Le requin et sa convergence

Une liste

* + *def buildHistogramme* prend un paramètre qui est **iteration** 
    - On cherche à déterminer la loi du requin. En lançant un nombre d’itérations il est possible d’en tirer une loi à partir des paramètres proposé par l’utilisateur
      * Paramètre
        + **iteration** : Le nombre d’itération souhaité

Doit être un entier

* + *def affichage\_requin* prend comme paramètre **which\_requin**
    - On représente notre requin ou sa convergence dans un graphique en mettant des informations pertinentes tels que le cas auquel il appartient, les paramètres utilisé …
      * Paramètre
        + **which\_requin** : Le requin que vous souhaitez afficher, soit le requin soit sa convergence.

Doit être une chaîne de caractère

* + *def affichage\_histogramme* ne prend pas de paramètre et ne renvoie rien.
    - On représente l’histogramme généré par le requin, qui va nous permettre de visualiser sa loi. Bien sûr nous mettrons de même manière des informations sur le graphique.