Projets

Vous trouverez ci-après la liste des sujets proposés pour réaliser vos projets d'informatique par groupes colles (donc issus du même groupe de TP). Les sujets sont de difficulté variée : plus il y a d'étoiles associées à un sujet, plus le sujet nous semble difficile (mais cette notion est subjective...). Chaque sujet présente un ou plusieurs objectifs à atteindre. Dans la description de chaque sujet, on vous propose parfois une façon de structurer les données ou de modulariser vos fonctions. Vous êtes libres de suivre ces propositions ou non. Enfin, sauf mention contraire, les seules bibliothèques permises sont celles du cours d'informatique. Si vous voulez en utiliser une autre, vous devez nous demander la permission de le faire.

Choix du sujet Vous devez faire parvenir à vos enseignants de TP, par courriel et au plus tard le vendredi 9 février 2024, à 20h, votre choix de sujets, classés par ordre de préférence, sous la forme :

```
choix 1 : sujet no xx (titre du sujet)
choix 2 : sujet no xx (titre du sujet)
choix 3 : sujet no xx (titre du sujet)
```

Nous arbitrerons ensuite l'attribution des sujets afin que deux groupes du même groupe de TP n'aient pas le même sujet.

Méthode de travail Dans la réalisation de votre projet, vous devrez être attentifs aux points suivants, qui simplifieront le déboguage de vos codes :

- on cherchera à modulariser au maximum son travail, c'est-à-dire à programmer un grand nombre de fonctions quitte à ce que chacune soit très courte. Pour fixer les idées, on pourra s'interdire toute fonction de plus de 50 lignes de code.
- Il est important de tester avec le plus d'esprit critique possible et dans un nouveau *shell* chaque fonction avant d'implémenter la suivante.
- Vous veillerez à sauvegarder régulièrement votre travail, en particulier chaque code fonctionnel avant d'essayer de l'améliorer ou lui ajouter des fonctionnalités.

Évaluation Une note sera attribuée à chaque groupe. Cette note tiendra compte :

- du travail global fourni pendant toute la durée du projet (chez vous et pendant la séance de suivi) ;
- des objectifs atteints (en fonction de la difficulté du projet) ;
- du rapport permettant de présenter le travail fait lors de votre projet, sur lequel vous devrez décrire les algorithmes implémentés (sans rentrer dans les détails du code), les problèmes rencontrés et les solutions trouvées ;
- du code python et de sa qualité. Merci de décrire dans le rapport comment exécuter le code.

Votre code python et votre rapport au format PDF sont à envoyer par courriel à vos enseignants.

Indice de difficulté Les sujets sont classés par indice de difficulté croissant :

- [⋆] Facile
- [⋆⋆] Moyen
- [⋆ ⋆ ⋆] Difficile
- [⋆ ⋆ ⋆⋆] Très difficile

Sujets proposés

Sujet n° 1

Somme des cubes [⋆]

Contexte : Pour tout entier $n \geq 1$, on définit $g(n)$ comme étant la somme des chiffres de n mis à la puissance 3.
ex : si $n = 214$, $g(n) = 2^3 + 1^3 + 4^3 = 73$
Objectif : Déterminer informatiquement quels sont les nombres entre 1 et 10 000 qui sont égaux à la somme des cubes de leurs chiffres.

Prolongement mathématique : Justifier que ce sont les seuls dans $\mathbb{N}^*$ *tout entier à vérifier cette propriété.*

Sujet n° 2

Temps d'attente avant rencontre de marches aléatoires [⋆]

Le but de ce projet est de confronter à une modélisation élémentaire l'adage selon lequel, lorsque deux personnes se cherchent, le mieux est que l'une d'entre elles reste sur place. On considère pour cela deux individus qui se déplacent aléatoirement dans un environnement confiné (d'abord un segment, puis un carré et enfin un cube), et on déterminera de manière empirique le temps moyen au bout duquel celles-ci se rencontrent. On considérera ensuite que l'un des individus reste sur place, et on comparera les valeurs obtenues.

Modélisation et Modularisation

- Chacun des individus partira d'une des extrémités du segment (ou du carré, du cube). On supposera que, à chaque instant, chaque individu se déplace d'un pas dans une direction choisie au hasard (on utilisera la bibliothèque `random`).
- On commencera par coder une fonction qui, partant de la position d'un individu à un instant donné, calcule sa position à l'instant suivant.
- On pourra tracer le graphe de l'espérance du temps d'attente avant rencontre en fonction de la longueur du segment (ou de l'aire du carré, du volume du cube) : est-ce qu'il y a corrélation linéaire ?
- Si le temps le permet, on pourra également considérer un cercle, sur lequel les individus effectuent à chaque étape une rotation d'angle $\pm\frac{2\pi}{n}$ avec n fixé. Là aussi on pourra tracer l'espérance en fonction de n.

Sujet n° 3

Nombres déficients/parfaits/abondants [⋆]

Contexte Pour tout entier $n \geqslant 2$, on définit $f(n)$ comme étant la somme des diviseurs de n (en comptant 1 mais pas n lui-même).

ex : les diviseurs de 6 *sont* 1, 2, 3 *et* 6 *donc* $f(6) = 1 + 2 + 3 = 6$.

Si $f(n) < n$, on dit que n est **déficient** ; si $f(n) = n$, on dit que n est **parfait** et enfin si $f(n) > n$, on dit que n est **abondant**.

Objectif Déterminer informatiquement

1. le plus petit nombre impair abondant ;
2. les 5 plus petits nombres pairs consécutifs abondants.

Prolongement : représenter graphiquement la proportion de nombres déficients/parfaits/abondants parmi les n premiers entiers, en fonction de n.

Sujet n° 4

Nombres premiers délicats [⋆]

Un nombre entier premier p est dit délicat si et seulement si en modifiant un seul de ses chiffres en base 10 on obtient un nombre qui n'est pas premier.

Par exemple, 13 n'est pas délicat car en modifiant son chiffre des unités, on peut obtenir 17 qui reste premier. On peut par contre vérifier que 294 001 est délicat car tout nombre obtenu en modifiant un de ses chiffres (par exemple 294 002, 294 011, 294 101…) n'est plus premier.

On désire écrire une fonction déterminant si un nombre premier est délicat, puis rechercher les plus petits nombres premiers délicats.

Objectifs

— Écrire une fonction déterminant si un entier n est un nombre premier délicat.

— Donner la liste de nombres premiers délicats inférieurs ou égaux à 10^6.

Sujet n° 5

Nombres premiers circulaires [⋆]

197 est appelé *nombre premier circulaire* car tous les nombres obtenus par rotations de ses chiffres (197, 719, 971) sont des nombres premiers.

Il y a 13 nombres premiers circulaires inférieurs à 100 : 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79 et 97.

Objectif Réaliser une fonction Python qui prend en paramètre un entier $N \geqslant 1$ et renvoie la liste des N plus petits nombres premiers circulaires.

Sujet n° 6

Traduction en rututu [⋆⋆]

Le rututu est un langage bizarre où la règle est que chaque syllabe en « u » doit être répétée. Par exemple, si l'on traduit en rututu la phrase :

mélusine a bu toute l'eau de la cruche.

cela donne :

mélulusine a bubu toute l'eau de la crucruche.

Plus précisément, on considère qu'une *syllabe en "u"* est une syllabe du type *consonne*-u (comme "bu"), ou *consonne-consonne*-u (comme "cru"), si le "u" est précédé de deux consonnes.

Pour simplifier, on considèrera des phrases sans majuscules.

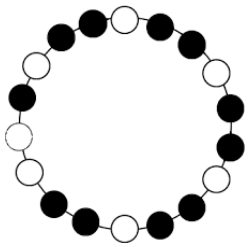
Objectif Réaliser une fonction Python qui prend en paramètre un texte, et le traduit en rututu.

Sujet n° 7

Colliers de perles binaires [⋆⋆]

On considère des colliers de perles constitués uniquement de perles blanches et noires. Mise à part leur couleur, les perles sont identiques.

On considère que deux colliers sont identiques si l'on peut obtenir l'un par une rotation de l'autre.



Objectif Écrire une fonction qui prend en paramètre un entier $n \in \mathbb{N}^*$ (en pratique, prenons $n \leqslant 12$, pour limiter les calculs) et qui renvoie le nombre de colliers possibles à n perles.

Sujet n° 8

Calcul de l'enveloppe convexe d'un nuage de points [⋆⋆]

L'objectif de ce projet est de calculer des enveloppes convexes de nuages de points dans le plan affine, un grand classique en géométrie algorithmique. On dit qu'une partie $\mathscr{C}$ du plan est convexe lorsque pour toute paire de points $M, N \in \mathscr{C}$, le segment de droite $[MN]$ est inclus dans $\mathscr{C}$. L'enveloppe convexe d'une partie $\mathscr{N}$, notée $\mathrm{Conv}(\mathscr{N})$, est la plus petite partie convexe contenant $\mathscr{N}$. Dans le cas où $\mathscr{N}$ est un ensemble fini (appelé nuage de points), le bord de $\mathrm{Conv}(\mathscr{N})$ est un polygone convexe dont les sommets appartiennent à $\mathscr{N}$, comme illustré dans la figure 1.

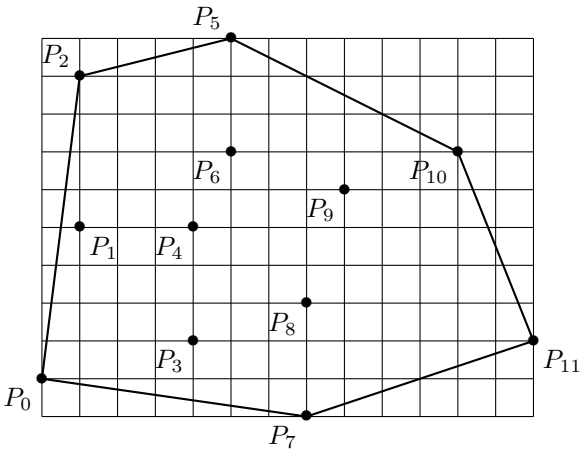


Figure 1

Le calcul de l'enveloppe convexe d'un nuage de points est un problème fondamental en informatique, qui trouve des applications dans de nombreux domaines comme :

- la robotique, par exemple pour l'accélération de la détection de collisions dans le cadre de la planification de trajectoire ;
- le traitement d'images et la vision, par exemple pour la détection d'objets convexes (comme des plaques minéralogiques de voiture) dans des scènes 2D ;
- l'informatique graphique, par exemple pour l'accélération du rendu de scènes 3D par lancer de rayons ;
- la théorie des jeux, par exemple pour déterminer l'existence d'équilibres de Nash ;
- la vérification formelle, par exemple pour déterminer si une variable risque de dépasser sa capacité de stockage ou d'atteindre un ensemble de valeurs interdites lors de l'exécution d'une boucle dans un programme ;

et bien d'autres encore.

Description de l'algorithme

Pour calculer l'enveloppe convexe d'un nuage de points, on peut suivre l'algorithme suivant.

- On commence par chercher un point du nuage qui est « sur le bord du nuage », comme par exemple le point P_0 de la figure 1.
- Ensuite, à partir de ce point on recherche le point « situé le plus à droite » de ce point (P_7 sur la figure 1).
- On cherche ensuite le point « situé le plus à droite » de P_7 ...et ainsi de suite jusqu'à revenir sur le point de départ.

Il faudra bien évidemment comprendre comment traduire mathématiquement le fait qu'un point est « situé le plus à droite » d'un autre point. On remarquera aussi que, si un point n'est pas « sur le bord du nuage » (par exemple le point P_4 sur la figure 1), alors aucun autre point n'est « situé le plus à droite » de celui-ci.

Modélisation et modularisation
Afin de faciliter les calculs, on pourra manipuler les points du nuage sous la forme de tableaux `numpy` de dimension 1, contenant les deux coordonnées.
Pour implémenter cet algorithme on pourra par exemple implémenter les fonctions suivantes.

- Une fonction qui génère un nuage de points aléatoires.
- Une fonction qui affiche le nuage de points.
- Une fonction qui détermine un point « sur le bord du nuage ».
- Une fonction qui détermine, à partir d'un point « sur le bord du nuage » le point « situé le plus à droite » de celui-ci.
- Une fonction qui affiche l'enveloppe convexe.

Sujet n° 9

Le jeu 2048 [⋆⋆]

Problématique Le but de se projet est de programmer le jeu 2048 afin de pouvoir y jouer de manière fonctionnelle.

On rappelle le principe général de ce jeu qui se déroule sur un plateau de 4 cases par 4 cases. Au début du jeu, deux cases contiennent aléatoirement le nombre 2 ou 4, les autres cases étant vides. À chaque étape de jeu :

1. on choisit une des quatre directions : haut, bas, gauche ou droite pour déplacer l'ensemble des cases dans la direction choisie ;
2. les cases contenant le même nombre fusionnent selon la direction considérée à l'étape précédente et leurs valeurs s'additionnent ;
3. une case vide est remplie aléatoirement par un 2 ou un 4.

Le jeu s'arrête lorsque 2048 est atteint.

Modélisation et modularisation

— On utilisera notamment le module `random` pour les choix à chaque étape.

— L'interaction avec l'utilisateur pourra se faire avec la commande `input` dans un premier temps : l'utilisateur donnera une des lettres `h,b,g,d` pour indiquer la direction à l'étape suivante.

— Le jeu doit indiquer lorsqu'il n'y a plus de mouvement possible (on veillera au fait que même lorsque la grille est pleine, une fusion peut être possible).

— À chaque étape, on doit obtenir une fenêtre graphique semblable à celle présentée sur le sujet (sans couleur dans un premier temps, puis en jouant sur les couleurs ensuite avec, par exemple, une couleur de plus en foncée puis affinée comme dans le jeu réel).

— On pourra finaliser le jeu en permettant à l'utilisateur d'utiliser directement les touches haut, bas, gauche et droite du clavier avec, par exemple, le module `tkinter`.

Sujet n° 10

Interrupteurs [⋆⋆]

Dans un bâtiment, $n+1$ pièces alignées (numérotées de 0 à n) sont chacune munie d'un interrupteur qui allume/éteint la lumière.

0	1	2	⋯	$n-1$	n

Initialement, toutes les pièces sont éteintes. On réalise alors les actions suivantes :

— On actionne (switch) tous les interrupteurs, instantanément, et on attend une minute.

— On actionne tous les interrupteurs des pièces multiples de 2, et on attend une minute.

— On actionne tous les interrupteurs des pièces multiples de 3, et on attend une minute.

— etc.

— On actionne tous les interrupteurs des pièces multiples de n, et on attend une minute.

Objectifs

1. écrire une fonction de n qui donne la liste des pièces éclairées à la fin du processus.
2. (si l'objectif 1 a été atteint) écrire une fonction de n qui dit quelle pièce entre $\frac{n}{2}$ et n a été le plus longtemps éclairée.

Sujet n° 11

Correcteur d'orthographe [⋆⋆]

Problématique Le but de se projet est de créer un correcteur d'orthographe élémentaire pour des fichiers texte. On basera ce correcteur sur le fichier `dico.txt` (à télécharger) contenant la liste des mots du dictionnaire.

Un mot mal orthographié est un mot n'appartenant pas à la liste du dictionnaire `dico.txt`. Dans le cas d'une coquille, on peut essayer d'identifier le mot qui se rapproche le plus du mot mal orthographié.

Pour cela, on peut utiliser ce qu'on appelle la distance de Levensthein entre deux chaînes de caractères A et B. Il s'agit du nombre d'opérations minimal pour aller de A à B où les opérations considérées sont :

— substitution d'un caractère de A en un caractère de B ;

— ajout dans A d'un caractère de B ;

— suppression d'un caractère de A.

Ceci permet de quantifier la distance entre deux mots en créant notamment un tableau de la forme suivante avec les chaînes de caractères `niche` et `chiens` (la méthode de création de ce tableau est disponible sur la page Wikipedia associée présentant la distance de Levensthein).

		C	H	I	E	N	S
	0	1	2	3	4	5	6
N	1	1	2	3	3	4	5
I	2	2	2	2	3	4	5
C	3	2	3	3	3	4	5
H	4	3	2	3	4	4	5
E	5	4	3	3	3	4	5

La distance de Levenshtein est ici égale à 5.

Modélisation et modularisation

— On dispose d'un fichier `texte_original.txt` qui contient potentiellement des fautes d'orthographe (celui-ci est encodé en `utf8`).

— On commencera par coder une fonction qui analyse si un mot (qui sera une chaîne de caractère en Python) fait partie du fichier `dico.txt`.

— On optimisera cette fonction afin qu'elle ne fasse pas de comparaison avec tous les mots du dictionnaire.

— On créera une fonction qui, à partir du fichier `texte_original.txt` crée un fichier `fautes.txt` contenant les mots identifié comme faux et le numéro de ligne où apparaît cette faute. On pourra utiliser la fonction `open` et la méthode `write`.

— On créera une fonction qui calcule la distance de Levensthein entre deux chaînes de caractères A et B.

— On créera une fonction qui parcourt `texte_original.txt`, identifie les mots potentiellement faux et propose une correction avec une distance de Levensthein minimale. Celle-ci interagit avec l'utilisateur avec une commande du type `input` où l'utilisateur valide ou non cette correction. La fonction créera deux fichiers `texte_corrige.txt` et `fautes.txt` contenant respectivement le texte avec les corrections validées ainsi que les fautes non corrigées comme dans la fonction précédente.

Sujet n° 12

Évolution de la répartition d'un caractère génétique au sein d'une population et sélection naturelle [⋆⋆]

On considère un caractère génétique α. Le but de ce projet est de déterminer quelle est la probabilité de disparition de α après un grand nombre de générations, et *a contrario* quelle est la probabilité qu'à terme tous les individus le possèdent, en fonction de la probabilité $p \in [0, 1]$ de transmission du gène. On pourra pour cela s'intéresser à différents modèles.

Modélisation et Modularisation

- On modélise la population par une liste, chaque coordonnée valant 1 si l'individu associé possède le caractère α et 0 sinon.
- Abstraction faite de la génétique Mendelienne, on peut considérer qu'un enfant a une probabilité p de posséder le caractère α si au moins l'un de ses parents le possède. Pour simplifier, on supposera que chaque couple engendre deux descendants (la taille de la population à chaque génération sera donc constante), et on fera abstraction de la notion de sexe (n'importe quel individu peut s'accoupler avec n'importe quel autre).
- On codera une fonction qui, étant donnée une liste décrivant la population à une génération donnée, renverra la liste décrivant la population à la génération suivante. On formera pour cela des couples suivant une loi uniforme (on utilisera la bibliothèque `random`).
- On pourra commencer l'étude dans le cas où, à l'instant initial, un seul individu possède le caractère α (on fera évoluer ce nombre initial si le temps le permet).
- On tracera le graphe représentant la probabilité d'extinction (après 1000 générations) de α en fonction de p, et la probabilité que tous les individus possède le caractère α en fonction de p.
- On pourra essayer de mettre en évidence l'équilibre de Hardy-Weinberg : la fréquence d'un génotype donné dans une population est constante.
- Dans un second temps on pourra supposer que le caractère α possède une influence se traduisant sur le nombre de descendants d'un couple dont l'un et/ou l'autre des membres le possède : pour modéliser cela, on supposera par exemple que ces couples ont trois descendants à chaque génération. Ces deux modèles devraient permettre de vérifier que la théorie de l'évolution peut être fondée sur la diversité génétique (résultant des mutations) au sein d'une population, ainsi que sur la sélection naturelle.

Sujet n° 13

Interférences quantiques photon par photon [⋆⋆]

On se propose de construire une animation simulant l'impact des photons arrivant un par un sur un écran où les franges d'interférence selon l'axe (Ox) sont sinusoïdales. L'écran a pour taille arbitraire $[-3,3]$ selon (Ox) et $[-1,1]$ selon (Oy) et il est constitué de 250 pixels. On fera arriver au maximum 100 000 photons.

Afin de construire cette simulation, il est nécessaire d'effectuer un tirage aléatoire :

— de différentes valeurs d'une grandeur X : $x_1, x_2, \ldots, x_N$, sachant que X possède une densité de probabilité continue P_X telle que $\mathrm{d}P_X = A\left(1+\cos(2\pi x)\right)\ \mathrm{d}x$;

— de différentes valeurs d'une grandeur Y : $y_1, y_2, \ldots, y_N$, sachant que Y possède une densité de probabilité continue P_Y uniforme.

Afin de déterminer ces différentes valeurs, on utilisera la méthode de la transformation réciproque. Soit la fonction de répartition de la variable aléatoire X suivant la loi que l'on veut simuler :

$$f_X(x) = P(X \leqslant x) = \int_{X_{\min}}^{x} \mathrm{d}P_X = \int_{X_{\min}}^{x} A\left(1+\cos(2\pi\xi)\right)\ \mathrm{d}\xi$$

Soit U la loi uniforme sur l'intervalle $[0,1]$. On cherche à déterminer la loi suivie par la variable aléatoire $f_X^{-1}(U)$. Pour cela on cherche sa fonction de répartition définie par $P\left(f_X^{-1}(U) \leqslant x\right)$ pour tout réel x. Comme f_X est croissante (propriété des fonctions de répartition) :

$$P\left(f_X^{-1}(U) \leqslant x\right) = P\left(U \leqslant f_X(x)\right) \tag{1}$$

Par définition de la fonction de répartition d'une loi uniforme, $P\left(U \leqslant f_X(x)\right) = f_X(x)$ donc (1) devient simplement $P\left(f_X^{-1}(U) \leqslant x\right) = f_X(x)$ ce qui revient à dire que la fonction de répartition de la variable aléatoire $f_X^{-1}(U)$ est $f_X(x)$. Par unicité de la fonction de répartition, la variable aléatoire $f_X^{-1}(U)$ suit la même loi que X.

Ainsi, pour appliquer la méthode de la transformation réciproque, il faut :

— tirer de manière aléatoire une valeur U_0 comprise entre 0 et 1 avec une loi de probabilité uniforme ;

— déterminer la variable aléatoire $x_0 = f_X^{-1}(U_0)$ dès lors que la fonction réciproque de f_X est connue ;

— ou chercher la variable aléatoire x_0 telle que $f_X(x_0) - U_0 = 0$.

Modélisation et Modularisation

— Afin de vérifier que vous avez compris la méthode de la transformation réciproque, l'appliquer dans le cas où la fonction de répartition de la variable aléatoire est une loi exponentielle du type $f(x) = 1 - \exp\left(-\lambda x\right)$ avec $\lambda > 0$.

— Afin de construire l'animation simulant l'impact des photons arrivant un par un sur un écran où les franges d'interférence selon l'axe (Ox) sont sinusoïdales :

  — Implémenter le tracé de la loi de probabilité et de la fonction de répartition associée pour tout $x \in [-3,3]$ et pour tout $y \in [-1,1]$ en effectuant le tracé distinct des 4 courbes sur une seule figure.

  — Tracer pour un nombre de photons égal à 1, 10, 100 et 1 000, la répartition sous forme d'histogramme selon (Ox) et (Oy) des impacts sur l'écran. Une méthode de résolution par dichotomie est attendue.

  — Vérifier que pour 1 000 photons, on retrouve les lois de probabilité imposées.

  — Afficher en 2D les points d'impacts pour 1, 10, 100 et 1 000 photons. Vérifier alors que pour 10 000 photons les franges d'interférences apparaissent.

  — Réaliser une animation permettant de visualiser au fur et à mesure l'impact des 100 000 photons.

Sujet n° 14

La bataille navale [⋆⋆]

Problématique
Le but de ce projet est de programmer un jeu de bataille navale, dans lequel on pourra défier l'ordinateur. On implémentera donc la stratégie suivie par celui-ci.

Modélisation et modularisation

- On utilisera (par exemple) une mer de taille 6×6. La flotte sera composée de vaisseaux de taille $2, 2, 3, 3, 5$.
- Il y a deux plateaux de jeu, qui sont des tableaux 2D contenant des 0 et des 1, mais qui se lisent différemment suivant qu'il représente ma flotte ou la flotte adverse :
  - pour ma flotte : chaque coefficient 0 signifie qu'il n'y a rien à cet endroit, chaque 1 qu'il y a un bateau à cet endroit (on mettra un 2 lorsque le bateau aura été touché à cet endroit), et chaque -1 qu'un coup a été tiré dans l'eau à cet endroit ;
  - pour la flotte adverse : chaque coefficient 0 signifie que cet endroit est inconnu, chaque coefficient 1 signifie qu'un bateau a été touché à cet endroit (on mettra un 2 lorsque le bateau aura été coulé complètement), et chaque coefficient -1 qu'un coup a été tiré dans l'eau à cet endroit.
- On implémentera d'abord un algorithme permettant de placer la flotte au hasard sur la mer. Pour cela, on étudiera les navires les uns après les autres. Pour chacun d'eux, on commence par choisir un terme du tableau et une direction au hasard (on utilisera la bibliothèque `random`), et on essaye de placer le navire à partir de ce terme et suivant cette direction : si cela est impossible (car on rencontre un autre navire ou le bord du plateau de jeu) alors on recommence.
- On écrit ensuite un code correspondant au déroulement du jeu. Celui-ci a pour fonction de :
  - positionner les flottes au hasard ;
  - donner alternativement la main au joueur (via la primitive `input`) et à la machine (en appelant la fonction définie ci-dessous) tant que personne n'a gagné la partie ;
  - à chaque fois que la main est donnée au joueur, afficher à l'écran ses deux plateaux de jeu afin qu'il puisse décider du coup suivant ;
  - afficher un message à chaque fois qu'un bateau est coulé.
- Enfin, on implémente la stratégie suivante pour la machine. Celle-ci commence par tirer au hasard dans la zone inconnue, et dès qu'elle a ainsi touché un bateau, elle tourne autour du point de touche pour trouver dans quelle(s) direction(s) il se prolonge, afin de couler définitivement le bateau (on utilisera donc une boucle).

Sujet n° 15

Le jeu du taquin [⋆⋆]

Le taquin est un jeu en forme de damier composé de 15 petits carreaux numérotés de 1 à 15 qui glissent dans un cadre prévu pour 16. Il consiste à remettre dans l'ordre les 15 carreaux à partir d'une configuration initiale quelconque.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

Jeu du taquin.

Objectifs
Ce projet consiste à créer une interface rudimentaire de ce jeu (à l'aide de `input` par exemple) puis à remplir les objectifs suivants.
Dans ce qui suit, une configuration initiale est dite résoluble si on peut remettre en place les 15 carreaux à partir de celle-ci. Toutes les configurations initiales ne sont pas résolubles.

— Écrire une fonction permettant de générer une configuration initiale résoluble aléatoire.

— Écrire une fonction permettant de déterminer si une configuration initiale donnée est résoluble ou pas (on pourra commencer par aller lire la page wikipédia du jeu du taquin qui décrit la condition pour qu'une configuration initiale donnée soit résoluble). Une fois cette fonction écrite, on pourra aussi essayer d'améliorer la fonction précédente.

— Écrire une fonction permettant de résoudre le jeu du taquin à partir d'une configuration initiale résoluble donnée. On pourra ensuite essayer d'optimiser l'algorithme de cette fonction afin d'effectuer le moins de glissements possibles.

— Enfin, si tous les objectifs précédents ont été remplis, on pourra essayer d'améliorer l'interface graphique et de remplacer les chiffres du taquin par une partie d'une image donnée.

Sujet n° 16

Mesure de la rotation différentielle du Soleil par observation des taches solaires [⋆ ⋆ ⋆]

Le but de ce projet est d'implémenter un algorithme permettant de mettre en évidence la rotation différentielle du Soleil. La vitesse angulaire du Soleil varie selon la latitude considérée car le Soleil n'est pas un corps solide. Pour le vérifier, on se propose, au travers de ce projet, d'observer les taches solaires à la surface du Soleil.

Environ tous les 11 ans, apparaissent à la surface du Soleil des taches sombres, appelées taches solaires, dont la durée de vie varie de un à quelques mois et qui résultent de la déformation de boucles du champ magnétique interne du fait de la rotation du Soleil. Le champ magnétique bloque le mouvement convectif de la matière et diminue les apports d'énergie, rendant ces zones plus froides et donc plus sombres.

Remarque : dans ce projet, on négligera la vitesse angulaire de rotation de la Terre autour du Soleil ainsi que sa vitesse de révolution.

Modélisation et Modularisation

— On commencera par récupérer une archive d'images mise à disposition.

— On codera une fonction qui renvoie le nombre, la longitude et la latitude des taches solaires sur chaque image.

— On tracera la courbe de la longitude d'une tache en fonction de la date d'observation dont on exploitera la pente.

— On codera une fonction permettant d'obtenir la pente et une estimation de son erreur. Pour ce faire, on utilisera la méthode des moindres carrés qui minimise la somme des écarts quadratiques :

$$\sum_{i=1}^{n} (\ell_i - a t_i - b)^2$$

où ℓ_i est la longitude de la tache solaire à la date t_i et $n \in \mathbb{N}$ le nombre d'images sur lesquelles apparaissent la tache. Les expressions de la pente de la droite a et de l'ordonnée à l'origine b sont respectivement :

$$a = \frac{\mathrm{Cov}\,(t, \ell)}{V(t)} \quad \text{et} \quad b = \bar{\ell} - a\bar{t}$$

où nous avons introduit les notations de moyenne, de covariance et de variance des séries considérées, respectivement définies comme :

$$\bar{t} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \bar{\ell} = \frac{1}{n}\sum_{i=1}^{n} \ell_i, \qquad \mathrm{Cov}\,(t, \ell) = \left(\frac{1}{n}\sum_{i=1}^{n} t_i\,\ell_i\right) - \bar{t}\,\bar{\ell} \quad \text{et} \quad V(t) = \mathrm{Cov}(t, t)$$

Pour évaluer la qualité de l'approximation, on utilisera le carré du coefficient de corrélation $R \in [-1; 1]$, défini comme :

$$R = \frac{\mathrm{Cov}(t, \ell)}{\sqrt{V(t)\,V(\ell)}}$$

Si $R = 0$ les deux variables ne sont pas corrélées mais si $R = \pm 1$, les deux variables sont linéairement dépendantes.

— On mettra en évidence la vitesse de rotation différentielle du Soleil.

Sujet n° 17

Ordre lexicographique [⋆ ⋆ ⋆]

On appelle *mot* toute chaîne de caractères n'utilisant que les 26 lettres de l'alphabet, en minuscules et sans accent. L'ordre lexicographique sur ces mots est l'ordre alphabétique. On dira alors qu'un mot `M1` est *inférieur* à un autre mot `M2` s'il apparaît avant dans l'ordre alphabétique.

Objectifs

1. écrire une fonction qui prend en paramètres deux mots et qui dit si le premier est inférieur au deuxième ;
2. (si l'objectif 1 a été atteint) écrire une fonction qui prend une liste de mots et qui la range dans l'ordre alphabétique.

Indication La fonction `ord`, qui prend en argument un **unique** caractère `chr` et qui renvoie le nombre représentant le code Unicode de ce caractère, pourra être utilisée.

Sujet n° 18

Réponses temporelles des SLCI [⋆ ⋆ ⋆]

On cherche à définir un code permettant de déterminer l'expression de la réponse temporelle d'un système, de fonction de transfert connue sous la forme d'une fraction rationnelle, soumis à une entrée excitatrice, de fonction temporelle connue.

Problématique
Il n'existe à l'heure actuelle aucune méthode triviale pour calculer la transformée de Laplace inverse d'une fraction rationnelle. Les méthodes les plus abouties utilisent des concepts avancés d'intégration dans $\mathbb{C}$ et des paramètres de tolérance machine toujours difficile à ajuster. On se propose ici de définir un code qui permet de déterminer l'expression exacte de sa réponse temporelle à partir de sa décomposition en éléments simples.

Modularisation

- On pourra commencer par définir des fonctions associées aux transformées de Laplace inverse des fonctions usuelles puis définir une fonction qui permet de réaliser leur assemblage.
- On cherchera ensuite à définir une fonction permettant de décomposer en éléments simples une fraction rationnelle donnée sous la forme d'un quotient de 2 polynômes en $p \in \mathbb{C}$.
  - Pour ce faire, on pourra commencer par déterminer les racines du polynôme du dénominateur et leur multiplicité afin de déterminer la forme de la décomposition. Pour déterminer les racines d'un polynôme, vous pourrez dans un premier temps utiliser la fonction `roots` de la bibliothèque `numpy`. Par exemple, pour $3,2x^2+2x+1$, les racines sur $\mathbb{C}$ sont obtenues avec la commande :

    ```
    racines = np.roots([3.2,2,1])
    ```

  - Une fois la forme de la décomposition trouvée, il s'agira de déterminer les n coefficients $x = (x_1, x_2, \dots) \in \mathbb{R}^n$ en résolvant un système linéaire de la forme $Ax = b$ où $A \in \mathcal{M}_{n,n}(\mathbb{R})$ est une matrice de facteurs réels et $b \in \mathbb{R}^n$ une matrice colonne, toutes deux issues de l'identification des numérateurs en puissance de p. Pour déterminer les coefficients, on pourra utiliser la fonction `solve` de la bibliothèque `numpy.linalg` sous la forme :

    ```
    coefficients = numpy.linalg.solve(A,b)
    ```

- Une fois la réponse décomposée dans le domaine de Laplace, il s'agira de déterminer les originales temporelles puis de les assembler pour obtenir l'expression temporelle de la réponse.
- Enfin, il suffira de simuler la réponse temporelle.

Sujet n° 19

Résolution d'un sudoku [⋆ ⋆ ⋆]

Le sudoku est un jeu consistant à remplir une grille de taille $(9, 9)$, divisée en 9 carrés de taille $(3, 3)$ avec les chiffres de 1 à 9 de sorte que dans chaque ligne, chaque colonne et chaque carré, tous les chiffres de 1 à 9 apparaissent exactement une fois.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Exemple de sudoku.

Description de l'algorithme
Pour résoudre un sudoku, on peut suivre l'algorithme suivant.

- On commence par choisir une case libre (on pourra dans un premier temps se contenter de choisir la première case libre rencontrée) et une valeur *a priori* possible pour cette case (c'est-à-dire telle que cette valeur n'apparait pas dans la même ligne, la même colonne ou le même carré que cette case) et on l'ajoute à la grille.
- On continue ce processus tant qu'il reste des cases libres.
- Si il reste une case libre et qu'aucune valeur n'est possible pour cette case, cela signifie qu'on a précédemment ajouté une valeur incorrecte à une case. Dans ce cas, on enlève la dernière valeur ajoutée et on essaye de trouver une autre valeur pour cette case. Si cela n'est pas possible, il faut supprimer la valeur précédente, etc.

Modélisation
Pour détecter rapidement les contraintes sur les valeurs des cases libres, on peut commencer par assigner à chaque case de la grille un numéro de 1 à 81. Ensuite, pour garder en mémoire ces contraintes, on pourra utiliser un tableau `numpy` `S` à trois dimensions de taille $(9, 9, 10)$ tel que :

- Pour tout $(i, j) \in [\![0, 8]\!]^2$, `S[i,j,0]` contient la valeur de la case de la i-ème ligne et la j-ème colonne (en ayant indexé les lignes et les colonnes de 0 à 8) ou bien la valeur 0 si celle-ci est libre.
- Pour tout $(i, j) \in [\![0, 8]\!]^2$ et tout $k \in [\![1, 9]\!]$. Si la case (i, j) est libre, `S[i,j,k]` contient le numéro d'une case empêchant la case (i, j) de prendre la valeur k. Par exemple, si la case $(0, 0)$ contient la valeur 7, alors la case $(2, 1)$ ne peut pas prendre la valeur 7. On retiendra en mémoire cette contrainte en stockant le numéro de la case $(0, 0)$ dans `S[2,1,7]`.
- Lorsqu'il y a plusieurs cases qui imposent une même contrainte, il n'est pas utile de garder en mémoire toutes ces cases. Seule la première contrainte sera utile.

Il pourra aussi être utile d'avoir en mémoire une liste `ListeCasesLibres` contenant le numéro des cases libres et une liste `ListeAffectation` gardant en mémoire la liste des affectations qui ont été effectuées, sous la forme d'un couple contenant le numéro de la case affectée et la valeur donnée à cette case.
À chaque étape de l'algorithme, il faudra faire attention à bien mettre à jour les données contenues dans le tableau `S` et les deux listes `ListeCasesLibres` et `ListeAffectation`.

Modularisation
Pour implémenter cet algorithme on pourra par exemple implémenter les fonction suivantes.

- Une fonction qui initialise les données à partir d'une grille de départ contenue dans un tableau `numpy` de taille $(9, 9)$.
- Une fonction qui affiche l'état du sudoku (on pourra simplement utiliser `matplotlib.pyplot` pour cela).

— Une fonction qui prend en entrée les indices de ligne et de colonne d'une case et qui renvoie son numéro, et la fonction réciproque de cette dernière.

— Une fonction qui ajoute une valeur dans une case libre (et qui met à jour les données).

— Une fonction permettant de revenir en arrière et de supprimer une valeur dans la grille (et qui met à jour les données).

— Une fonction qui détermine une case libre.

— Une fonction qui détermine une valeur possible pour une case libre.

— Une fonction qui implémente l'algorithme principal.

Optimisation

Une fois que l'algorithme ci-dessus sera implémenté, on pourra :

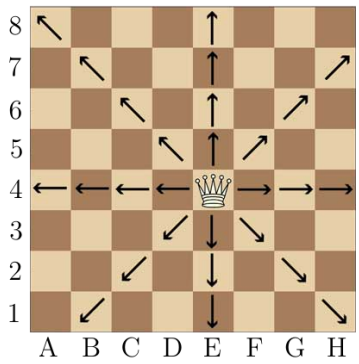
— essayer d'optimiser l'algorithme (par exemple en choisissant de manière plus judicieuse la prochaine case libre à affecter) ;

— améliorer l'interface graphique.

Sujet n° 20

Dames en prise [⋆ ⋆ ⋆]

Aux échecs, la position d'une pièce est donnée par ses coordonnées : une lettre entre A et H pour le numéro de colonne, et un chiffre entre 1 et 8 pour le numéro de ligne. La pièce la plus puissante, la dame, peut prendre les pièces ennemies qui se trouvent sur une même ligne/colonne qu'elle, ou sur une même diagonale. Un problème classique est d'arriver à placer n dames sur un échiquier sans qu'aucune d'entre elles ne soit menacée de prise par une autre.



Objectif On cherche le nombre de configurations à n dames sans prise et le nombre maximum de dames que l'on peut placer sur l'échiquier.

Modélisation et modularisation

— Écrire une fonction prenant comme arguments les coordonnées de deux dames (par exemple `'B6','F5'`) et qui dit si oui ou non une menace l'autre.

— En déduire pour toutes les positions de dames possibles, la liste des cases libres pour poser une autre dame.

— Écrire une fonction permettant de placer une nouvelle dame sans qu'elle soit en prise avec celles déjà sur l'échiquier.

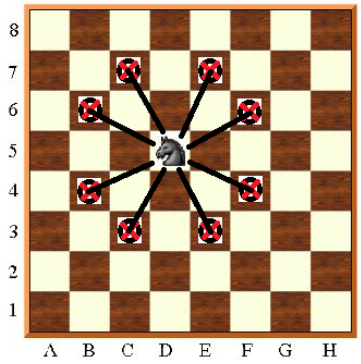
— Déterminer l'ensemble des configurations possibles, si besoin par parcours récursif.

— Déterminer le nombre maximum de dames sans prises qu'il est possible de placer et les nombre de configurations correspondant.

Sujet n° 21

Cavaliers en prise [⋆ ⋆ ⋆ ⋆]

Aux échecs, la position d'une pièce est donnée par ses coordonnées : une lettre entre A et H pour le numéro de colonne, et un chiffre entre 1 et 8 pour le numéro de ligne. L'un de ces pièces, le cavalier, peu prendre des pièces ennemies en réalisant un mouvement en forme de L : 2 cases dans une direction, puis 1 case dans une direction orthogonale. Un problème classique est d'arriver à placer n cavaliers sur un échiquier sans qu'aucun d'entre eux ne soit menacé de prise par un autre.



Objectif On cherche le nombre de configurations à n cavaliers sans prise et le nombre maximum de cavaliers que l'on peut placer. Sur Wikipédia il est dit qu'il est possible d'en disposer 32.

Modélisation et modularisation

— Écrire une fonction prenant comme arguments les coordonnées de deux cavaliers (par exemple `'B6','F5'`) et qui dit si oui ou non un menace l'autre.

— En déduire pour toutes les positions de cavaliers possibles, la liste des cases libres pour poser un autre cavalier.

— Écrire une fonction permettant de placer un nouveau cavalier sans qu'il soit en prise avec ceux déjà sur l'échiquier.

— Déterminer l'ensemble des configurations possibles à $n = 3$, 4 ou 5 cavaliers, si besoin par parcours récursif.

— Essayer de déterminer le nombre maximum de cavaliers que l'on peut placer sur l'échiquier et le nombre de configurations associées. On fera attention à la complexité du problème...