

Objectifs de la séance : apprendre à utiliser SageMath dans un contexte de simulations aléatoires.

Ouvrir un terminal, puis lancer Sage avec la commande `sage -n jupyter`. Créer une feuille de calcul SageMath. Pour exécuter le contenu d'une cellule de la feuille de calcul, taper (Shift+Enter). Le résultat du dernier calcul est automatiquement stocké dans la variable `_`; par exemple, la commande `type(_)` affiche le type de ce résultat. Une aide thématique est disponible à l'adresse :

<http://doc.sagemath.org/html/en/reference/>.

Opérations, variables, méthodes. On commence par introduire les bases de Python et de Sage.

- (1) Calculer $3 + 2 + 5$, $\sin(\frac{\pi}{3})$, $\sqrt{1 - \sin^2(\frac{\pi}{3})} - \cos(\frac{\pi}{3})$. Toutes les fonctions usuelles sont déjà programmées!
- (2) On stocke ou modifie une variable nommée x avec la commande `x = val`, où `val` est la valeur que l'on veut donner à x . Recalculer la valeur de $\sqrt{1 - \sin^2(\frac{\pi}{3})} - \cos(\frac{\pi}{3})$ en stockant d'abord une variable égale `x` à $\sin(\frac{\pi}{3})$. Pour afficher la valeur d'une variable `x`, on utilise la commande `print(x)`, ou simplement `x`.
- (3) La documentation d'une fonction `func` est donnée par la commande `func?`. Demander la documentation de la fonction `sin`, et celle de `matrix`. Créer une variable `M` égale à la matrice $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$.
- (4) Le type d'un objet `x` dans Sage (ou Python) est donné par la fonction `type(x)`. Quel est le type de `3`? de `3.1`? de la matrice créée à la question précédente? On peut forcer le type d'un nombre `x` (réel, entier, rationnel) avec les commandes `RR(x)`, `ZZ(x)` et `QQ(x)`. Créer trois versions différentes du nombre `3`.
- (5) Tous les objets dans Sage ont des méthodes qui leurs sont associées, et qui dépendent du type de l'objet en question. Une méthode `method` d'un objet `x` est appelée par la commande `x.method()`. Pour avoir la liste des méthodes associées à un objet `x`, on peut taper `x. +` (Tab). Calculer le déterminant, les valeurs propres et l'inverse de la matrice `M`.
- (6) Presque tous les objets mathématiques sont déjà implémentés dans Sage. En utilisant la documentation, réaliser les tâches suivantes :
 - (a) Créer un graphe (Graph) qui est un triangle, et l'afficher.
 - (b) Obtenir la table de multiplication du corps fini (Finite...) \mathbb{F}_4 à 4 éléments.
 - (c) Calculer le développement de Taylor à l'ordre 6 de la fonction $\exp(1 - \cos(x))$.

Fonctions, listes, itération. On détaille maintenant quelques techniques de programmation plus avancées : fonctions, listes, boucles `for`.

- (1) Une fonction $f(x)$ d'une variable x est définie en Sage par la commande :


```
def func(x):
    instructions
    return (le resultat)
```

Attention, tout le contenu de la fonction doit être indenté dans le code. Créer une fonction $g(x) = \sqrt{1 - \exp(-x)}$, et calculer $g(0.3)$. Afficher son graphe sur l'intervalle $[a, b] = [0, 2]$ en utilisant la commande `plot(lambda x : g(x), x, a, b)`.

- (2) On peut tester une condition, par exemple dans les instructions d'une fonction avec :

```
if condition:
    instructions_1
else:
    instructions_2
```

De nouveau, les instructions doivent être indentées par rapport au reste du programme pour être reconnues comme telles. Les conditions que l'on peut tester sont l'égalité ($a == b$), l'inégalité stricte ($a < b$) ou large ($a <= b$), etc. Créer une fonction

$$h(x) = \begin{cases} e^{-x} & \text{si } x \geq 0, \\ 1 & \text{si } x < 0, \end{cases}$$

et afficher son graphe sur $[-2, 2]$.

- (3) Une liste d'objets (a, b, c) est codée par la commande `[a, b, c]`. On appelle le i -ième élément d'une liste L par `L[i]` (les éléments sont numérotés à partir de 0), et on ajoute un élément x à la fin d'une liste L avec `L.append(x)`. Créer une fonction `liste_factorielle(n)` qui renvoie la liste des entiers $k!$, avec k entre 1 et n (indication : on peut appeler une fonction f récursivement à l'intérieur d'elle-même).
- (4) Si l'on a des opérations à effectuer sur les éléments d'une liste L , on utilise :

```
for x in L:
    instructions
```

La liste des entiers entre 1 et n est donnée par `range(1, n+1)` ou par `[1..n]`. Créer une fonction `factoriel(n)` qui calcule $n!$.

Variables aléatoires. On explique comment travailler avec des variables aléatoires.

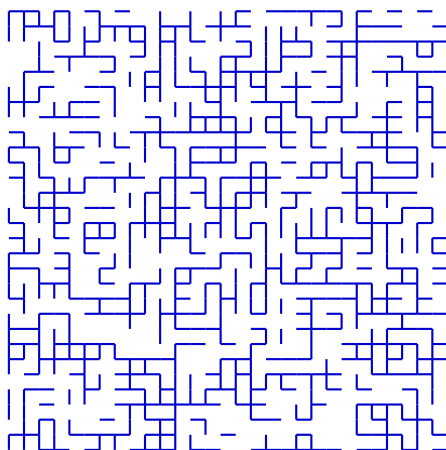
- (1) Une variable uniforme sur $[0, 1]$ est donnée par la commande `random()`. À partir de celle-ci, on peut construire n'importe quelle variable réelle X de fonction de répartition F_X , car si $U \sim \mathcal{U}([0, 1])$, alors $F_X^{-1}(U) \sim X$. Créer :
- (a) une fonction `random_exp(t)` qui renvoie une variable de loi exponentielle, dont la densité est $1_{x>0} e^{-tx} t dx$;
 - (b) une fonction `random_bernoulli(p)` qui renvoie une variable de Bernoulli de paramètre p .
 - (c) et une fonction `random_binomial(n, p)` qui renvoie une variable binomiale de paramètres n et p .
- (2) En fait, la plupart des lois continues sont déjà implémentées dans Sage, et on peut y accéder avec la commande `RealDistribution(type, parametres)`. Ainsi,
- ```
U = RealDistribution("uniform", [0, 1])
G = RealDistribution("gaussian", 1)
```
- définissent des générateurs aléatoires de lois respectives  $\mathcal{U}([0, 1])$  et  $\mathcal{N}(0, 1)$ . Si  $X$  est un générateur aléatoire défini dans Sage, on obtient des nombres aléatoires indépendants avec la commande `X.get_random_element()`. Utiliser les méthodes du générateur gaussien  $G$

défini ci-dessus pour dessiner la densité et la fonction de répartition de la loi gaussienne standard, et pour obtenir une liste de 10 nombres aléatoires gaussiens indépendants.

- (3) La fonction `sage.plot.histogram.histogram(L)`, qu'on peut évaluer en une liste `L`, dresse un histogramme des valeurs de la liste. Dresser un histogramme de 10000 valeurs du générateur aléatoire gaussien (indication : on peut rajouter après l'argument `L` la commande `,bins=n` dans les parenthèses pour avoir un histogramme en  $n$  colonnes).
- (4) Vérifier la loi des grands nombres et le théorème central limite pour des sommes de variables aléatoires de Bernoulli indépendantes (indication : `sum(L)` renvoie la somme d'une liste).
- (5) Dans Sage, les représentations graphiques peuvent être construites avec des listes d'instructions. On initialise un graphique avec la commande `G = Graphics()`, et on rajoute un objet `Obj` sur le graphe (ligne, polygone, texte, *etc.*) avec la commande `G += Obj`. On affiche ensuite `G` avec `G.show()`, avec éventuellement des options dans l'argument de `show()`. Par exemple,

```
G = Graphics()
G += line([(0,0),(0,1)])
G += line([(1,0),(0,1)])
G += line([(0,0),(1,0)])
G.show(axes=False,aspect_ratio=1)
dessine un triangle rectangle isocèle.
```

Écrire une fonction `Percolation(n,p)` qui garde chaque arête de la grille  $\llbracket 0, n \rrbracket \times \llbracket 0, n \rrbracket$  avec probabilité  $p$  indépendamment pour chaque arête, et affiche le résultat.



**Chaînes de Markov.** On programme diverses chaînes de Markov : marches aléatoires, processus de branchement, *etc.*

- (1) Écrire un programme `Marche(n,p)` qui affiche les  $n$  premières étapes d'une marche aléatoire sur  $\mathbb{Z}$  de paramètre  $p$ . Vérifier expérimentalement que si  $p > \frac{1}{2}$ , alors la marche aléatoire tend presque sûrement vers  $+\infty$ ; tandis que si  $p = \frac{1}{2}$ , alors la marche semble revenir infiniment souvent en 0.
- (2) L'anneau  $\mathbb{Z}/N\mathbb{Z}$  est défini dans Sage par `R = Integers(N)`, et un entier  $k$  modulo  $N$  est alors appelé par `R(k)`. Par exemple,

```
R = Integers(9)
R(4) + R(7)
```

renvoie 2 (ou plus précisément, la classe dans l'anneau quotient  $\mathbb{Z}/9\mathbb{Z}$  qui contient 2). Écrire un programme `MarcheCercle(N, n)` qui calcule les  $n$  premières étapes d'une marche aléatoire sur  $\mathbb{Z}/N\mathbb{Z}$ , issue de 0 et avec probabilités de transition

$$P(k, k) = P(k, k - 1) = P(k, k + 1) = \frac{1}{3}.$$

Vérifier avec un histogramme que pour  $n$  grand, la loi de la position  $X_n$  est quasi-uniforme sur  $\mathbb{Z}/N\mathbb{Z}$ .

- (3) Soit  $\mu$  une mesure de probabilité sur  $\mathbb{N}$ . Le processus de Galton-Watson de loi  $\mu$  est la suite aléatoire  $(X_n)_{n \in \mathbb{N}}$ , telle que  $X_{n+1}$  est la somme de  $X_n$  variables aléatoires indépendantes de loi  $\mu$ . Il représente l'évolution d'une population dont chaque individu a aléatoirement  $Y \sim \mu$  enfants, indépendamment pour chaque individu. Calculer la matrice de transition de cette chaîne de Markov, et écrire un programme `GaltonWatson(Y, n, k)` qui dépend d'un générateur aléatoire  $Y$  et donne les  $n$  premières générations du processus, partant de  $X_0 = k$ .
- (4) Programmer une classe `ChaineMarkov` qui s'initialise à partir d'une liste d'états et d'une matrice de transition, et qui contient des méthodes :
- pour calculer une trajectoire de la chaîne de Markov, et la mesure empirique associée ;
  - pour calculer la mesure invariante de la chaîne.