FEUILLE DE TD/TP 5
SIMULATION DE CHAÎNES DE MARKOV ET MÉTHODES MCMC

1. Simulation de chaînes de Markov.

Exercice 1. Le marché des forfaits de téléphonie mobile d'un pays se répartit entre trois entreprises A, B et C. Chaque mois, une certaine proportion des clients change d'opérateur et cette proportion est supposée invariante dans le temps. Ainsi modélise-t-on l'évolution du choix d'un client par une chaîne de Markov $(X_n)_{n\in\mathbb{N}}$ dont la matrice de transition est présentée ci-dessous :

$$P = \begin{pmatrix} 0.8 & 0.05 & 0.15 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix}.$$

(1) Montrer que P est récurrente, irréductible et apérodique.

On note $E = \{A, B, C\}$. La matrice de transition P est irréductible car pour tout $(x, y) \in E$, $P(x, y) > 0$. Comme l'espace d'état E est fini, on en déduit que tous les états sont récurrents, donc P est récurrent. Comme P est irréductible et récurrente, tous les états ont la même période, et on a $P^1(A, A) = 0.8 > 0$ donc $\text{pgcd}\{n \in \mathbb{N} : P^n(A, A) > 0\} = 1$, ce qui justifie que A est de période 1. On en déduit que P est apériodique.

(2) Écrire un programme qui prend en paramètre une loi μ sur $\{1, \dots, n\}$ et qui simule une réalisation de μ.

```
def real(mu):
    #La loi mu est traitée comme un np.array contenant les probabilités de chaque éléments entre 1 et n
    F = np.cumsum(mu) #On calcule la fonction de répartition de mu
    u = np.random.rand() #On tire un nombre aléatoire entre 0 et 1
    return np.argmax(F > u) + 1  #On calcule Q(u) = inf{i \in {0,..,N-1} : F[i] > u} + 1
```

(3) Écrire un programme qui prend en paramètre une loi μ et un entier n et qui simule les $(n+1)$ premiers termes de la chaîne $(X_k)_{0\leq k\leq n}$ avec loi initiale μ. Représenter l'évolution de X_k en fonction du temps.

```
#Simulation de la trajectoire de la chaine de Markov de matrice de transition P, de loi initiale mu, de longueur n.
def simu_X(mu,P,n):
    X = np.zeros(n+1)
    X[0] = real(mu)
    for i in range(1,n+1):
        X[i] = real(P[int(X[i-1])-1,:])
    return X

P = np.array([[0.8,0.05,0.15],[0.1,0.7,0.2],[0.1,0.1,0.8]])
mu = np.array([0.1,0.2,0.7])
simu_X(mu,P,10)

#Représentation graphique
def graph(mu,P,n):
    X = simu_X(mu,P,n)
    plt.figure()
    plt.plot(X)
    plt.show()

graph(mu,P,10)
```

(4) Expliquer pourquoi la chaîne de Markov admet une unique probabilité invariante π. La calculer numériquement.

La chaîne de Markov est irréductible et récurrente, d'espace d'état fini. Elle est donc récurrente positive, ce qui permet d'affirmer qu'elle admet une unique probabilité invariante π. Par définition, π

vérifie l'égalité $\pi P = \pi$: en notant $\pi = \begin{pmatrix} x & y & z \end{pmatrix}$, on a donc

$$\begin{pmatrix} x & y & z \end{pmatrix} \begin{pmatrix} 0.8 & 0.05 & 0.15 \\ 0.1 & 0.7 & 0.2 \\ 0.1 & 0.1 & 0.8 \end{pmatrix} = \begin{pmatrix} x & y & z \end{pmatrix},$$

ce qui équivaut au système linéaire suivant, d'inconnues x, y et z :

$$\begin{cases} -0.2x + 0.1y + 0.1z &= 0 \\ 0.05x - 0.3y + 0.1z &= 0 \\ 0.15x + 0.2y - 0.2z &= 0. \end{cases}$$

Une résolution par pivot de Gauss permet de montrer que l'ensemble des solutions de ce système est de la forme $\{(\frac{8}{11}z, \frac{5}{11}z, z); z \in \mathbb{R}\}$. Comme π doit être une probabilité, ses coordonnées doivent également vérifier l'équation $x + y + z = 1$, ce qui se traduit, d'après ce qui précède, par $\frac{8}{11}z + \frac{5}{11}z + z = 1$, donc $z = \frac{11}{24}$. On en déduit finalement que $\pi = \begin{pmatrix} \frac{1}{3} & \frac{5}{24} & \frac{11}{24} \end{pmatrix}$.

On peut également proposer une méthode numérique :

```
#pi(P) détermine la probabilité invariante associée à la matrice de transition P.
def pi(P):
    n = P.shape[0]
    A = np.transpose(P) - np.eye(n)
    A[0,:] = np.ones(n)
    b = np.zeros(n)
    b[0] = 1
    return np.linalg.solve(A,b)
```

(5) On suppose qu'à l'instant initial, l'opérateur du client est choisi selon la loi $\mu_0 = (0.6, 0.4, 0)$. Quelle est la limite presque sûre de $\frac{1}{n+1}\sum_{k=0}^{n} \mathbb{1}_{X_k=A}$? Illustrer cette convergence numériquement.

Comme la chaîne est récurrente positive, on a d'après le théorème ergodique

$$\frac{1}{n+1}\sum_{k=0}^{n} \mathbb{1}_{X_k=A} \xrightarrow[n\to\infty]{} \pi(A) \quad \mathbb{P}_{\mu_0}\text{-p.s.}$$

```
#Vérification numérique
mu_zero = np.array([0.6,0.4,0])
n = 10000
print(1/(n+1) * np.sum(simu_X(mu_zero,P,n) == 1).astype(int))
```

Exercice 2. On considère la marche aléatoire simple au plus proche voisin sur $\mathbb{Z}$ issue de $x \in \mathbb{Z}$, $(S_n)_{n\in\mathbb{N}}$ définie par

$$\begin{cases} S_0 = x \\ S_{n+1} = S_n + X_{n+1} \end{cases}$$

où $(X_n)_{n\in\mathbb{N}}$ est une suite de v.a.i.i.d. telle que

$$\mathbb{P}(X_1 = 1) = p \quad \text{et} \quad \mathbb{P}(X_1 = -1) = 1 - p$$

pour un certain $p \in]0, 1[$.

(1) Écrire une fonction `ma(x,p,N)` qui produit une trajectoire $(S_n)_{0\le n\le N}$ de la marche issu de `x` et de paramètre `p`.

```
def rademacher(p):
    u = np.random.rand()
    if u < p:
        return 1
    else:
        return -1

def ma(x,p,N):
    trajectory = np.zeros(N+1)
    trajectory[0] = x
    for i in range(1,N+1):
        trajectory[i+1] = trajectory[i] + rademacher(p)
    return trajectory
```

(2) Écrire une fonction `maT(a,b,x,p)` qui produit une trajectoire $(S_n)_{0\le n\le T_{ab}}$ de la marche issue de $a \le x \le b$ jusqu'au temps d'atteinte de $\{a, b\}$.

```
def maT(a,b,x,p):
    trajectory = np.array([x])
    state = x
    while state != a and state != b:
        state = state + rademacher(p)
        trajectory = np.append(trajectory,state)
    return trajectory
```

(3) Utiliser la fonction précédente pour construire une fonction `sortie(a,b,x,p,m)` qui estime la probabilité que la marche issue de `x` touche `a` avant `b` en simulant `m` réalisations de $(S_n)_{0\leq n\leq T_{ab}}$ et une fonction `EspT(a,b,x,p,m)` qui estime l'espérance $\mathbb{E}_x[T_{ab}]$.

```
def sortie(a,b,x,p,m):
    result = np.zeros(m)
    for i in range(m):
        state = x
        while state != a and state != b:
            state = state + rademacher(p)
        if state == a:
            result[i] = 1
    mean = np.mean(result)
    std = np.std(result)
    icsize = 1.96 * std / np.sqrt(m)
    return np.array([mean,std,mean - icsize, mean + icsize])

def EspT(a,b,x,p,m):
    result = np.zeros(m)
    for i in range(m):
        state = x
        while state != a and state != b:
            state = state + rademacher(p)
            result[i] += 1
    mean = np.mean(result)
    std = np.std(result)
    icsize = 1.96 * std / np.sqrt(m)
    return np.array([mean,std,mean - icsize, mean + icsize])
```

Exercice 3. On considère la marche aléatoire simple au plus proche voisin sur $\mathbb{Z}^2$ issue de $(0;0)\in\mathbb{Z}^2$, $(S_n)_{n\in\mathbb{N}}$ définie par

$$\begin{cases} S_0 = (0;0) \\ S_{n+1} = S_n + X_{n+1} \end{cases}$$

où $(X_n)_{n\in\mathbb{N}}$ est une suite de v.a.i.i.d. telle que

$$\mathbb{P}(X_1=(1;0))=p_1,\ \mathbb{P}(X_1=(-1;0))=p_2,\ \mathbb{P}(X_1=(0;1))=p_3,\ \text{ et } \mathbb{P}(X_1=(0;-1))=p_4\,.$$

où $\sum_{i=1}^4 p_i = 1$.

(1) Écrire une fonction `maZ2(p,N)` qui produit une trajectoire $(S_n)_{0\leq n\leq \mathtt{N}}$ de la marche issue de `x` et de paramètre $p=(p_1,\dots,p_4)$.

(2) Représenter graphiquement les trajectoires de la marche dans $\mathbb{Z}^2$ et illustrer le fait que $(0;0)$ est récurrent ssi pour tout $1\leq i\leq 4$, $p_i = 1/4$.

Exercice 4. *File d'attente.*
On étudie l'évolution d'une file d'attente à un guichet. On suppose qu'un client est servi par unité de temps. On note N_n le nombre de clients arrivant dans la $n^{\text{ième}}$ unité de temps. On suppose que les variables N_n s'écrivent $G_n - 1$ où les $(G_n)_{n\in\mathbb{N}}$ sont des v.a.i.i.d. de loi géométrique $\mathcal{G}(p)$ pour un certain $p\in]0;1[$ et qu'un client arrivant à la période n ne peut pas être servi avant la période $n+1$. Finalement, on note X_n le nombre de clients dans la file d'attente à l'instant n et on suppose que $X_0 = 0$.

$$\forall n\in\mathbb{N},\ X_{n+1} = N_{n+1} + X_n - \mathbb{1}_{X_n\geq 1}\,.$$

(1) Écrire une fonction `File` qui simule le nombre de personnes dans la file d'attente sur n pas de temps pour un paramètre $p\in]0;1[$ fixé.

(2) Illustrer numériquement le fait que $(X_n)_{n\in\mathbb{N}}$ tend vers l'infini si $p<1/2$ et que 0 est un état récurrent de la chaîne si $p>1/2$.

2. Algorithme de Metropolis-Hastings

Exercice 5. *Loi de Poisson.*
En utilisant le noyau de transition $Q(x,\cdot) = \frac{1}{2}\delta_{(x-1)\vee 0} + \frac{1}{2}\delta_{x+1}$, écrire un programme qui simule une loi de Poisson de paramètre λ par l'algorithme de Metropolis-Hastings.

```
def Q(x,y):
    return 0.5*(max(x-1,0) == y).astype(int) + 0.5*(x+1 == y).astype(int)

def Pi(x,lambd):
    return np.exp(-lambd)*lambd**x/np.math.factorial(x)

def MetropolisHastings(lambd,N):
    x = 0
    for i in range(1,N):
        u = np.random.rand()
        y = (u < 0.5).astype(int)*(max(x-1,0)) + (u >= 0.5).astype(int)*(x+1)
        alpha = min(1,(Pi(y,lambd)*Q(y,x))/(Pi(x,lambd)*Q(x,y)))
        u = np.random.rand()
        if(u < alpha):
            x = y
    return x
```

Exercice 6. *Modèle d'Ising.*
Le modèle d'Ising est un modèle utilisé en physique statistique pour modéliser différents phénomènes, notamment le ferromagnétisme, dans lesquels des effets collectifs sont produits par des interactions locales entre particules à deux états. Nous allons ici étudier une version simple en dimension 2. On considère, pour un N fixé, le carré $C_N = \{0,\dots,N\}^2$. On définit l'espace des configurations $E = \{-1,1\}^{C_N}$ et on introduit sur cet espace la mesure de probabilité suivante appelée loi de Boltzmann :

$$\forall x \in E,\ \pi(x) = \frac{1}{Z_T} e^{-\frac{1}{T}H(x)} \quad \text{où} \quad H(x) = \frac{1}{2} \sum_{\substack{m,m' \in C_N \\ |m-m'|=1}} |x(m) - x(m')|^2$$

et Z_T est une constante de renormalisation pour que π soit bien une mesure de probabilité.
Le carré C_N modélise un réseau bidimensionnel d'atomes m et les quantités $x(m)$, que nous appelerons *spins* dans la suite de l'exercice, représentent une quantité physique lié à chaque atome ou à chaque sommet du réseau et ne pouvant prendre que deux valeurs, par exemple un moment magnétique positif ou négatif ou même la présence ou l'absence d'un atome à cet emplacement du réseau. On choisit pour simplifier les deux valeurs possibles égales à -1 et +1 d'où la définition de l'espace E. Le point essentiel du modèle est que les spins de tous ces atomes ne sont pas indépendants mais interagissent avec leurs voisins. L'idée de Boltzmann est de voir la loi de l'ensemble des spins comme une fonction décroissante de l'énergie du réseau $H(x)$. Intuitivement, moins les valeurs des $x(m)$ pour des m proches sont similaires, plus l'énergie du système est élevée et moins la configuration est probable.

Il est difficile de simuler une loi de Boltzmann, car on ne peut pas s'y prendre atome après atome, étant donnée la structure de dépendance interatomes. L'idée est alors d'utiliser un algorithme MCMC tel que la méthode de Metropolis-Hastings. On étudie ici le cas de deux noyaux de transition.

(1) On note x^m la configuration $x \in E$ où l'on a inversé le spin m :

$$x^m(k) = \begin{cases} x(k) & \text{si } k \neq m \\ -x(m) & \text{si } k = m \end{cases}$$

On considère alors le noyau de transition Q sur E défini par

$$\forall m \in C_N,\ Q(x, x^m) = \frac{1}{|C_N|}\ .$$

(a) Expliquer le mécanisme probabiliste associé à Q et montrer que Q vérifie les hypothèses nécessaire pour mettre en place l'algorithme de Metropolis-Hastings.

Pour tout $x \in E$, $Q(x,\cdot)$ est une distribution uniforme sur l'ensemble des "voisins proches" de x, c'est-à-dire les configurations qui ne diffèrent de x que par un spin. En particulier, $Q(x,\cdot)$ attribue une probabilité nulle sur tous les autres éléments de E. Notons que si x et y sont deux éléments de E qui diffèrent de $k \in \{0,\dots,N^2\}$ spins, il est ainsi possible de relier x avec y en allant de "voisin proche" en "voisin proche" en au moins k étapes en inversant les spins différents un par un, i.e $Q^k(x,y) = 1/|C_N|^k > 0$, donc Q est irréductible. Finalement, pour tout $x, y \in E$, on a $Q(x,y) > 0$ si et seulement si x et y ne diffèrent que d'un spin, si et seulement si $Q(y,x) > 0$. Donc Q vérifie bien les hypothèses nécessaires pour mettre en place l'algorithme de Métropolis-Hasting.

(b) Calculer $H(x^m) - H(x)$.

Par définition,

$$H(x) - H(x^m) = \frac{1}{2} \sum_{\substack{k,k' \in C_N \\ |k-k'|=1}} \left(|x(k) - x(k')|^2 - |x^m(k) - x^m(k')|^2 \right).$$

Pour tout $k, k' \neq m$, on a $|x(k) - x(k')|^2 = |x^m(k) - x^m(k')|^2$, donc il ne reste dans la somme que m et ses voisins, c'est-à-dire que

$$H(x) - H(x^m) = \frac{1}{2} \sum_{\substack{k \in C_N \\ |k-m|=1}} \left(|x(k) - x(m)|^2 - |x^m(k) - x^m(m)|^2 \right).$$

D'autre part, pour $|k-m| = 1$ on a $x^m(k) = x(k)$ et $x^m(m) = -x(m)$ donc

$$|x(k) - x(m)|^2 - |x^m(k) - x^m(m)|^2 = |x(k) - x(m)|^2 - |x(k) + x(m)|^2 = -4x(k)x(m),$$

et on en déduit que

$$H(x) - H(x^m) = -2x(m) \sum_{\substack{k \in C_N \\ |k-m|=1}} x(k).$$

(c) Programmer une fonction qui permet de simuler une réalisation de la loi π par l'algorithme de Metropolis-Hastings et le noyau Q.

```
def Ising_Naif(N,T,M):
    #Initialisation
    x = np.zeros((N,N))
    for i in range(0,N):
        for j in range(0,N):
            x[i,j] = np.random.choice([-1,1])
    #Simulation
    for k in range(M):
        y = x
        i = np.random.randint(0,N)
        j = np.random.randint(0,N)
        y[i,j] = -y[i,j]
        #On considère que les bords sont périodiques pour simplifier
        alpha = min(1,np.exp(2/T * x[i,j] * (x[(i+1)%N,j] + x[(i-1)%N,j] + x[i,(j+1)%N] + x[i,(j-1)%N])))
        u = np.random.rand()
        if(u < alpha):
            x = y
    return x
```

(2) On se propose de construire un noyau $\widetilde{Q}$ plus performant. Pour cela, on partitionne l'espace C_N en :

$$C_N^+ = \{ (m_1, m_2) \in C_N,\ m_1 + m_2 \text{ est pair} \} \quad \text{et}$$
$$C_N^- = \{ (m_1, m_2) \in C_N,\ m_1 + m_2 \text{ est impair} \}$$

Et pour $x \in E$, on note $x^+ = (x(m),\ m \in C_N^+)$ et $x^- = (x(m),\ m \in C_N^-)$. Ainsi $x = (x^+, x^-)$.

(a) Montrer qu'il existe une constante $C > 0$ telle que, pour tout $x \in E$,

$$\pi(x) = \frac{1}{C} \prod_{m \in C_N^+} \exp\left(\frac{2}{T}\, x(m) \cdot \sum_{\substack{m' \in C_N^- \\ |m-m'|=1}} x(m') \right)$$

(b) Soit $X = (X^+, X^-)$ une variable aléatoire de loi π. Montrer que, conditionnellement à $X^- = x^-$ les variables $X^+(m), m \in C_N^+$ sont indépendantes de loi respective :

$$\mathbb{P}(X^+(m) = 1 | X^- = x^-) = \frac{e^{-\frac{2}{T}M(x^-,m)}}{e^{-\frac{2}{T}M(x^-,m)} + e^{\frac{2}{T}M(x^-,m)}}$$

et

$$\mathbb{P}(X^+(m) = -1 | X^- = x^-) = \frac{e^{\frac{2}{T}M(x^-,m)}}{e^{-\frac{2}{T}M(x^-,m)} + e^{\frac{2}{T}M(x^-,m)}}$$

où

$$M(x^-, m) = \sum_{\substack{m' \in C_N^- \\ |m-m'|=1}} x^-(m') \, .$$

(c) On construit $\widetilde{Q}$ comme le noyau de transition associé à la chaîne de Markov suivante. On part de $X_0 = x_0$ et, pour construire X_{n+1} à partir de X_n, on tire une variable U de loi $\mathcal{U}([0,1])$, indépendamment de $X_1, \ldots, X_n$.

- Si $U \leq 1/2$, on tire Y^+ de loi $\pi(\cdot | X_n^-)$ et on pose $(X_{n+1}^+, X_{n+1}^-) = (Y^+, X_n^-)$.
- Si $U > 1/2$, on tire Y^- de loi $\pi(\cdot | X_n^+)$ et on pose $(X_{n+1}^+, X_{n+1}^-) = (X_n^+, Y^-)$.

Expliciter le noyau $\widetilde{Q}$ et montrer que $\widetilde{Q}$ vérifie les hypothèses de l'algorithme de Metropolis-Hastings.

(d) Utiliser ce nouveau noyau pour programmer une fonction qui permet de simuler une réalisation de la loi π par l'algorithme de Metropolis-Hastings.