2022

Exercice 1

On pose la matrice $A_{n,i}$ qui est constituée de $n-1$ colonnes u sauf la $i-$ième colonne qui est la vecteur v avec $u = \begin{pmatrix} 1 \\ \vdots \\ n \end{pmatrix}$ et $v = \begin{pmatrix} n \\ \vdots \\ 1 \end{pmatrix}$.

Non donné par l'énoncé Par exemple $A_{3,3} = \begin{pmatrix} 1 & 1 & 3 \\ 2 & 2 & 2 \\ 3 & 3 & 1 \end{pmatrix}$.

1. Écrire une fonction python qui renvoie $A_{n,i}$.

2. Quels sont les spectres de matrices $A_{n,i}$ pour n variant de 3 à 7 et i variant de 1 à n.

3. Quelle valeur propre α est commune à toutes les matrices $A_{n,i}$ pour $n \geq 3$ et $i \in [\![1; n]\!]$.

4. Soit $M \in \mathcal{M}_n(\mathbb{R})$. Montrer que M et M^T ont le même spectre.
Montrer que $\dfrac{n(n+1)}{2}$ est une valeur propre commune des $A_{n,i}$.

5. Montrer que $Sp(A_{n,i}) = \left\{\alpha, \beta_{n,i}, \dfrac{n(n+1)}{2}\right\}$, déterminer $\beta_{n,i}$ et montrer que $\beta_{n,i} < \dfrac{n(n+1)}{2}$.

Exercice 2

1. Écrire une fonction python qui renvoie $A \mapsto \displaystyle\sum_{k=0}^{+\infty} \frac{A^k}{k!}$.

2. Écrire une fonction python qui renvoie

$$A \mapsto \begin{cases} e^{\lambda}(A - \lambda I_n) & \text{si } Sp(A) = \{\lambda\} \\ \dfrac{1}{\lambda_1 - \lambda_2}\left(e^{\lambda_1}(A - \lambda_2 I_n) + e^{\lambda_2}(A - \lambda_1 I_n)\right) & \text{si } Sp(A) = \{\lambda_1, \lambda_2\} \text{ avec } \lambda_1 \neq \lambda_2 \end{cases}$$

Faire un test avec $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$, $B = \begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix}$ et $C = \begin{pmatrix} 2 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

Quelle conjecture peut-on faire ?

3. On prend $\lambda_1, \ldots, \lambda_s$ s scalaires différents 2 à 2 et $m_1, \ldots, m_s$ s entiers 2 à 2 différents et on note $r = \displaystyle\sum_{i=1}^{+\infty} m_i$.
Montrer que g est un isomorphisme avec

$$g : \begin{cases} \mathbb{C}_{r-1}[X] & \to & \mathbb{C}^r \\ S(X) & \mapsto & (S(\lambda_1), S'(\lambda_1), \ldots, S^{(m_1-1)}(\lambda_1), \ldots, S(\lambda_s), \ldots, S^{(m_s-1)}(\lambda_s)) \end{cases}$$

44.1. Montrer qu'il existe n matrices $F_{i,j}$ telles que

$$\forall A \in \mathcal{M}_n(\mathbb{C}), \qquad S(A) = \sum_{i=1}^{s} \sum_{j=0}^{m_i-1} S^{(j)}(\lambda_i) F_{i,j}.$$

4.2. d'autres questions

Exercice 3 N étudiants passent un examen. Ceux qui ne l'ont pas réussi doivent le repasser jusqu'à ce qu'ils réussissent. On suppose que la probabilité de réussir l'examen est constante égale à $p \in]0,1[$ à chaque passage et que les différents résultats sont indépendants. On pose $q = 1-p$.
On note X le nombre total d'examens passés, Y le nombre total de sessions et, pour tout $k \in [\![1,n]\!]$, X_k le nombre de fois où l'étudiant k passe l'examen.

1. Écrire une fonction Python permettant de simuler l'expérienxe et l'utiliser pour calculer l'espérance de X et celle de Y.
2. Quelle est la loi suivie par X_k ? Donner son espérance et sa variance.
3. Décrire X en fonction des X_k. En déduire sa loi, son espérance et sa variance. Vérifier la cohérence avec la question 1.
   (On fera les calculs avec $p = 1/3$ et $N = 3$.)
4. Que vaut $P(Y \leq n)$?
   On définit $f_N(x) = \sum_{k=0}^{N-1} q^x(1-q)^x$. *** $f_N(x) = \sum_{k=0}^{N-1} q^k(1-q^k)^x$

5.1. Justifier l'existence de $\int_0^{+\infty} f_N(t)dt$.

5.2. Via une comparaison série-intégrale, montrer que $f_N \sim -\frac{\ln q}{\ln N}$. ***

Exercice 4 On considère une population de m individus qui choisissent de voter pour deux candidats A et B. Seulement, avant les élections, les individus parlent entre eux et échangent de leurs convictions politiques donc chaque jour, un individu de la population (tiré au hasard) discute avec un autre (aussi tiré au hasard) et si ce dernier est d'un bord politique différent du premier, alors il est convaincu et change d'intention de vote. On note X_n le nombre d'individus qui comptent voter pour A le n-ème jour. Initialement, il y a a individus qui comptent voter pour A et $m-a$ individus pour B. Donc la probabilité initiale est presque sûre $P(X_0 = a) = 1$.
On notera $\pi_{n,k} = P(X_n = k)$ la probabilité que k personnes comptent voter pour A le n-ème jour.

1. Écrire une fonction qui prend n, m et a et qui renvoie le nombre de votants pour A au n-ème jour. Exécuter la fonction pour un grand intervalle de temps. Que remarque-t-on ?
2. Pour $k \in [\![0,m]\!]$, montrer que

$$P_{(X_n=k)}(X_{n+1} = k+1) = P_{(X_n=k)}(X_{n+1} = k-1) = \frac{k(m-k)}{m(m-1)}$$

   et $P_{X_n=k}(X_{n+1} = k) = 1 - \dfrac{2k(m-k)}{m(m-1)}$.
3. Pour $k \in [\![1,m-1]\!]$, montrer que

$$\pi_{n+1,k} = \frac{k(m-k)}{m(m-1)}\pi_{n,k-1} + \left(1 - \frac{2k(m-k)}{m(m-1)}\right)\pi_{n,k} + \frac{k(m-k)}{m(m-1)}\pi_{n,k+1}.$$

En déduire que $\pi_{n,k} \leq \left(\dfrac{m(m-1)-2}{m(m-1)}\right)^n$.

On définit les événements V_A : à partir d'un certain temps, tout le monde deut voter pour A et V_n symétriquement pour B.

4. Montrer que $P(V_A) = \lim\limits_{n\to+\infty} P(X=n)$ et que $P(V_B) = \lim\limits_{n\to+\infty} P(X=0)$.
Expliquer pourquoi $P(V_A)+P(V_B)=1$. Est-ce cohérent avec le comportement remarqué en 1 ?

5. Écrire une fonction qui teste V_A en prenant en entrée n, m et a et en renvoyant 1 si tout le monde finit par voter pour A et 0 sinon.

6. d'autres questions

2021

Exercice 5 Centrale Python 2021

Pour $u=(a,b,c)\in\mathbb{R}^3$, on définit la matrice $M_u = \begin{pmatrix} a & b & c \\ c & a & b \\ b & c & a \end{pmatrix}$.

11.1. Montrer que M_u s'écrit comme combinaison linéaire de trois matrices dont l'une est le carré de l'une des deux autres. En déduire un programme de paramètre une liste $[a,b,c]$ qui renvoie la matrice M_u.
Calculer $M_{(7,2,6)} \times M_{(-14,15,3)}$ et $M_{(-14,15,3)} \times M_{(7,2,6)}$.

1.2. Quelle est la structure de $\mathcal{M} = \{M_u, u\in\mathbb{R}^3\}$?
$\mathcal{M}$ est-il stable pour la loi $\times$? Pour $(A,B)\in\mathcal{M}^2$, a-t-on $A\times B = B\times A$?

22.1. Montrer que M_u est semblable à $\begin{pmatrix} a+b+c & 0 & 0 \\ 0 & a-\dfrac{1}{2}(b+c) & \dfrac{\sqrt{3}}{2}(b-c) \\ 0 & -\dfrac{\sqrt{3}}{2}(b-c) & a-\dfrac{1}{2}(b+c) \end{pmatrix}$.
Vérifier ce résultat avec le logiciel sur la matrice $M_{(1,2,3)}$.

2.2. Déterminer une condition nécessaire et suffisante portant sur $u=(a,b,c)$ pour que M_u soit diagonalisable dans $\mathbb{R}$.

3. On pose $S=\{(a,b,c)\in\mathbb{R}^3; a^2+b^2+c^2=1\}$, $T=\{(a,b,c)\in\mathbb{R}^3; a+b+c=1\}$ et $T'=\{(a,b,c)\in\mathbb{R}^3; a+b+c=-1\}$.
On note $U = S\cap(T\cup T')$. On définit le polynôme $P_m(X)=X^3-X^2+m$ où m désigne un réel.

3.1. Représenter P_m pour $m\in\{-0,1;0;0,1;0,2\}$ dans $\left[-1;\dfrac{3}{2}\right]$. Déterminer une condition nécéssaire et suffisante sur m pour que P_m admette trois racines réelles (éventuellement confondues).

3.2. Montrer que M_u est une matrice orthogonale si, et seulement si, $u\in U$.

3.3. Montrer que M_u est la matrice d'une rotation si, et seulement si, a,b et c sont les racines de P_m avec $m\in\left[0,\dfrac{4}{27}\right]$.

3.4. Caractériser géométriquement M_u pour $u=\left(\dfrac{2}{3},\dfrac{2}{3},-\dfrac{1}{3}\right)$.

Exercice 6 Centrale Python 2021

On appelle matrice à spectre diagonal les matrices pour lesquelles le polynôme caractéristique est scindé dans $\mathbb{R}$ et est de la forme $\chi_A = \prod_{i=1}^{n}(X - a_{ii})$ avec $A = (a_{ij})_{1\leq i,j\leq n}$.

1. Montrer que si une matrice a un spectre diagonale, alors elle est trigonalisable.

2. Ecrire une fonction Python qui prend en argument une matrice $A = (a_{ij})$ et qui renvoie le polynôme $P = \prod_{i=1}^{n}(X - a_{ii})$.
Calculer le polynôme caractéristique de quelques matrices 2×2 et 3×3 (des matrices qu'il n'a plus en tête) et en déduire si elles sont à spectre diagonal.

3. On note $\mathcal{E}_n$ l'ensemble des matrices à spectre diagonal. Caractériser $\mathcal{E}_2$.

4. Soit $M = \begin{pmatrix} A & B \\ 0 & C \end{pmatrix} \in \mathcal{M}_n(\mathbb{R})$ avec A et C deux matrices carrées.
Montrer que si A et C sont à spectre diagonal, alors M aussi.

5. Donner une matrice 4×4 à spectre diagonal contenant 13 coefficients non nuls.

6. Soit $M \in \mathcal{E}_n \cap \mathcal{A}_n$ où $\mathcal{A}_n$ est l'ensemble des matrices antisymétriques.

6.1. Calculer χ_M. Donner M^n.

6.2. Montrer que M^2 est diagonalisable et en déduire que $M^2 = 0$.

7. Deux autres questions

Exercice 7 Centrale Python 2021

On pose Σ l'ensemble des suites de la forme $(u_1, u_2) \in \mathbb{R}^2$ et $\forall k \geq 3$, $u_k = \frac{1}{k}(2u_{k-1} + (k-2)u_{k-2})$.

1.1. Prouver que Σ est un sous-espace vectoriel réel. Donner sa dimension.

1.2. On note $u(a, b)$ la suite de Σ avec $u(a, b)_1 = a$, $u(a, b)_2 = b$.
Que vaut $u(a, a)$?
En déduire comment décrire Σ complétement.

1.3. Ecrire une fonction Python $u(a, b, n)$ qui renvoie $u(a, b)_n$.

1.4. On pose $\forall n \in \mathbb{N}^*$, $A_n = \sum_{k=1}^{n-1} \frac{1}{k}$.
Après avoir écrit une fonction $A(n)$, calculer pour $n \in [\![2; 20]\!]$ $u(1, 3/2, n) - 2A(n)$.
En regardant le résultat, quelle conjecture peut-on faire ?

1.5. Montrer que $\forall (a, b) \in \mathbb{R}^2$, $u(a, b)$ est bornée.

2. On pose $f_u(x) = \sum_{k=1}^{+\infty} u_k x^k$ avec $u \in \Sigma$.

2.1. Montrer que le rayon de convergence de f_u est strictement positif.

2.2. Montrer que f_u est solution de l'équation différentielle

$$(x^2 - 1)f_u'(x) - 2f_u(x) = g(u_1, u_2)x + h(u_1, u_2)$$

avec g et h des fonctions en u_1 et u_2 à déterminer.
En déduire, sans calcul, que $x \mapsto \frac{x}{1-x}$ est fonction solution de cette équation.

2019

Exercice 8 Saint-Cyr 2019 **Python**

Soient $f : \left\{ \begin{array}{ccc} [0;1] & \to & \mathbb{R} \\ x & \mapsto & \frac{3}{2}x - x^3 \end{array} \right.$ et une suite $(u_n)_{n\in\mathbb{N}}$ vérifiant $\forall n \in \mathbb{N}, \quad u_{n+1} = \frac{3}{2}u_n - u_n^3$.

1. Avec Python, conjecturer le comportement de $(u_n)_{n\in\mathbb{N}}$ en fonction de $u_0 \in [0;1]$. Représenter la courbe représentative de f et la droite $y = x$ sur un même schéma, ainsi que $n \mapsto u_n$ pour différentes valeurs de u_0.
2. Pour $u_0 \in \left]0; \frac{1}{\sqrt{2}}\right[$, démontrer la conjecture de la première question.
3. Factoriser $X^3 - \frac{3}{2}X + \frac{1}{\sqrt{2}}$.
4. Montrer que $\frac{1}{\sqrt{2}} - u_{n+1} \underset{n\to+\infty}{\sim} \frac{3}{\sqrt{2}}\left(u_n - \frac{1}{\sqrt{2}}\right)^2$.

Exercice 9 Centrale 2019 **Python**

On définit la suite de fonctions $(u_n)_{n\in\mathbb{N}}$ par, $\forall a \in \mathbb{R}$, $u_0(a) = a$ et $\forall n \in \mathbb{N}$, $u_{n+1}(a) = \arctan\left(\frac{u_n(a)}{1+\sqrt{1+u_n^2(a)}}\right)$.

1. Écrire une fonction *suite*(N,a) qui donne les valeurs de $u_n(a)$ pour tout $n \in [\![0; N]\!]$. Que renvoie *suite*(3,0) ? *suite*(10,2018) ?
2. Conjecturer la limite, lorsque $n \to +\infty$, de la suite $(u_n(a))$. Le vérifier pour $a \in \{1, 4, 10, 100\}$. On note C_1 cette conjecture. Conjecturer également la limite de la suite $(2^n u_n(a))$. On note C_2 cette nouvelle conjecture.
3. Tracer le graphe de la fonction $2^{10}u_{10}$ sur l'intervalle $[-30; 30]$ avec un pas de $0,01$.
4. Démontrer, pour tout $x \in \mathbb{R}$, $|\arctan x| \leq |x|$. Démontrer la conjecture C_1.
5. Déterminer la nature des séries de terme général $u_n(a)$, $u_n^2(a)$, $\ln\left(\frac{2u_{n+1}(a)}{u_n(a)}\right)$.
6. Soit g définie par $g(a) = \sum_{n=0}^{+\infty} u_n(a)$. Montrer que g est continue sur $\mathbb{R}$.

Exercice 10 Saint-Cyr 2019

Soit $f : \left\{ \begin{array}{ccc} \mathbb{R}_+ & \to & \mathbb{R} \\ x & \mapsto & x^2 - 2 \end{array} \right.$. L'objectif est de calculer une valeur approchée de $\sqrt{2}$.

1. Montrer qu'il existe un unique $a \in \mathbb{R}_+$ tel que $f(a) = 0$.
2. Avec Python, calculer a avec la méthode de Newton, en détaillant cette méthode.
3. Montrer que cette méthode conduit à une suite (x_n) vérifiant $\forall n \in \mathbb{N}$, $x_{n+1} = \frac{x_n^2 + 2}{2x_n}$. Dans la suite, on prendra $x_0 = 1$.
4. Montrer que cette suite est bien définie et que, $\forall n \in \mathbb{N}$, $x_n \geq 1$.
5. Montrer que $\forall n \in \mathbb{N}$, $|x_n - \sqrt{2}| \leq \frac{1}{2^{2^n}}$.

6. Définir le nombre de décimales exactes qu'on peut obtenir en évaluant x_5 puis x_10.

7. Modifier le programme afin de prendre en compte la variable m pour obtenir une valeur approchée de a à moins de 10^{-m} par la méthode de Newton.

Exercice 11 Centrale 2019-Python

Soit $u \in \mathcal{L}(\mathbb{R}^n)$ et $v \in \mathbb{R}^n$.

On dit que v est un vecteur de Krylov pour u si la famille $(v, u(v), u^2(v), \cdots, u^{n-1}(v))$ est une base de $\mathbb{R}^n$.

11.1. Écrire une fonction prenant en argument une liste de n vecteurs et vérifiant si cette famille est une base de $\mathbb{R}^n$ ou non.

1.2. Écrire une fonction `estKrylov(M,v)` d'arguments une matrice M de $\mathcal{M}_n(\mathbb{R})$ et un vecteur v de $\mathbb{R}^n$, indiquant si v est un vecteur de Krylov associé à M.

2. Soit v un vecteur de Krylov associé à u.
Donner la forme de la matrice de u dans la base $(v, u(v), u^2(v), \cdots, u^{n-1}(v))$.

3. Soit $P = a_0 + a_1X + \cdots + a_dX^d$ un polynôme non nul annulateur de u.
Montrer que la famille $\big(v, u(v), u^2(v), \cdots, u^d(v)\big)$ est liée.

4. En déduire que si u est diagonalisable et possède une valeur propre de multiplicité supérieure ou égale à 2, alors il n'existe pas de vecteur de Krylov pour u.

Exercice 12 Saint-Cyr 2019

On considère l'équation différentielle : (E) : $\quad (x - e^{-x})y' + (1 + e^{-x})y = 1$.

On pose $g : x \mapsto x - e^{-x}$.

1. Montrer que g s'annule en un unique point que l'on notera α.

2. Écrire un programme Python pour trouver α à 10^{-4} près.
On note (a, b) des réels tels que $a < \alpha < b$ et $|b - a| \leq 10^{-4}$. On note aussi $I_1 = [0; a]$ et $I_2 = [b; +\infty[$.

3. Sur Python représenter la solution de (E) sur I_1 sachant que $y(0) = 1$.

4. De même sur I_2 avec $y(1) = 1$.

5. Déterminer une solution de (E) sur $[\alpha; +\infty[$.

Exercice 13 Saint-Cyr 2019

On considère une urne contenant 2 boules blanches et 1 boule noire. On note X la variable aléatoire qui correspond à l'instant où on tire deux boules blanches consécutivement.

On note $u_n = P(X = n)$.

1. Écrire une fonction **selection()** qui renvoie 1 si la boule tiré est blanche et 0 si elle est noire.

2. Écrire une fonction **simulation()** qui renvoie le numéro n dy tirage pour lequel on a 2 boules blanches à la suite.

3. Donner une liste statistique pour 10^4 tirages des u_n pour $n \in [\![2, 20]\!]$.

4. Montrer que $u_{n+2} = \dfrac{1}{3}u_{n+1} + \dfrac{2}{9}u_n$.

5. Donner une expression de u_n.

Exercice 14 Saint-Cyr 2019

On considère deux urnes A et B. À l'instant initial l'urne A contient p particules et B est vide. À chaque instant (discrétisé) une particule choisie aléatoirement passe d'une uren à l'autre. Soit T une variable aléatoire modélisant le premier instant auquel on reient à l'état initial.

1. Montrer que l'on ne peut revenir à l'état initial que pour des instants pairs. Calculer $P(T=2)$, $P(T=4)$.
2. On modélise l'état des particules à un instant donné par une liste L de longueur p où chaque élément vaut 1 si la particule correspondante est dans A, 0 si elle est dans B.
   Écrire une fonction Python ehrenfest(L) qui modélise le passage de l'instant n à l'instant $n+1$. On pourra utiliser la fonction randint(p) importée du module random.
3. Écrire une fonction Python experience(p) qui modélise l'expérience avec p particules et renvoie le premier instant où on retrouve l'état initial.
4. Écrire une fonction Python moyenneT(n,p) renvoyant la valeur moyenne de T pour n expériences avec p particules.
   Tracer la courbe de la valeur moyenne de T en fonction de p.
   Quelle conjevture faire sur la limite de T quand $p \to +\infty$?
5. Quel phénomène thermodynamique est modélisé par cette expérience ?

Exercice 15 Centrale 2019

On pose $F : x \mapsto \displaystyle\int_0^{+\infty} \frac{e^{-xt}}{1+t}dt$.

1. Tracer le graphe de la fonction F avec python et conjecturer le domaine de définition de F, son signe, ses variations, sa limite en $+\infty$.
2. Démontrer vos conjectures.
3. Démontrer que F est continue.
4. Montrer que F est solution d'une équation différentielle du premier ordre à coefficient constant. En déduire que F est de classe C^∞.
   à l'oral : résoudre cette équation différentielle.
5. Montrer que $F(x) = e^{\pm x}\displaystyle\int_0^{+\infty} \frac{e^{\pm t}}{t}dt$. signe à retrouver

2018

Exercice 16 Centrale 2018-Python

Soit $n \in \mathbb{N}^*$, on considère

$$N_n = \begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ \vdots & & & \ddots & 1 \\ 0 & \cdots & \cdots & \cdots & 0 \end{pmatrix} \in \mathcal{M}_n(\mathbb{R}) \text{ et } R_n = \begin{pmatrix} 1 & \frac{1}{2} & -\frac{1}{8} & \cdots & \frac{(-1)^{n-1}}{n2^{2n-1}}\binom{2n-2}{n-1} \\ 0 & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & -\frac{1}{8} \\ \vdots & & & \ddots & \frac{1}{2} \\ 0 & \cdots & \cdots & 0 & 1 \end{pmatrix} \in \mathcal{M}_n(\mathbb{R}).$$

11.1. Écrire une fonction en python qui renvoie N_n à partir de la donnée de $n \in \mathbb{N}^*$.

1.2. Calculer $(N_n)^n$ pour $n \in \{2; 5; 10\}$. Quelle conjecture peut-on faire ?

1.3. Démontrer cette conjecture. On dit que N_n est une matrice nilpotente.

22.1. Écrire une fonction en python qui renvoie R_n à partir de la donnée de $n \in \mathbb{N}^*$.

2.2. Calculer $(R_n)^2$ pour $n \in \{2; 5; 10\}$. Quelle conjecture peut-on faire ?

33.1. Soit $n \in \mathbb{N}^*$, montrer qu'il existe un polynôme P_n tel que :

$$\forall x \in]-1; 1[, \qquad \sqrt{1+x} = P_n(x) + \underset{n\to+\infty}{O}(x^n).$$

Préciser ses coefficients et son degré.

3.2. Montrer que $(P_n(X))^2 - X - 1$ est divisible par X^n.

4. Démontrer la conjecture faite en 2.2.

2017

Exercice 17 ENSAM 2017 **Python**

1. Déterminer une valeur approchée de $\sqrt{2}$ par une méthode dichotomique
(on pourra utiliser l'équation $x^2 - 2 = 0$)

2. Soit la suite $(u_n)_{n\in\mathbb{N}}$ définie par $u_0 > 0$ et $u_{n+1} = \dfrac{1}{2}\left(u_n + \dfrac{2}{u_n}\right)$. Écrire une fonction qui calcule u_n.

3. On admet que $\left|u_n - \sqrt{2}\right| \leq \left(\dfrac{u_0 - \sqrt{2}}{2\sqrt{2}}\right)^{2n}$.

Écrire une nouvelle fonction donnant une valeur approchée de $\sqrt{2}$ et comparer les deux résultats obtenus.

4. Justifier que (u_n) converge vers $\sqrt{2}$.

Exercice 18 Centrale 2017-Python

Soit $P \in \mathbb{C}_p[X], Q \in \mathbb{C}_q[X]$ et u définie sur $\mathbb{C}_{q-1}[X] \times \mathbb{C}_{p-1}[X]$ par $u(A, B) = AP + BQ$.
Soit $\mathcal{B} = ((1,0), (X,0), \cdots, (X^{q-1}, 0), (0,1), (X,1), \cdots, (X^{q-1}, 1), \cdots (0, X^{p-1}), (X, X^{p-1}), \cdots, (X^{q-1}, X^{p-1}))$ une base de $\mathbb{C}_{q-1}[X] \times \mathbb{C}_{p-1}[X]$

1. Donner la matrice $M_{P,Q}$ de u dans $\mathcal{B}$ en fonction des coefficients de P et Q.

2. Écrire un programme en python qui prend les listes associées aux coefficients de P et de Q en paramètre et donne la matrice $M_{P,Q}$.

3. On choisit $P(X) = X^4 + X^3 + 1$ et $Q(X) = X^3 - X + 1$. Montrer que :
$\exists!(A_0, B_0) \in \mathbb{C}_2[X] \times \mathbb{C}_3[X]$ tel que $A_0P + B_0Q = 1$.

4. On choisit $P(X) = (X-1)(X-2)(X+2)$ et $Q_a(X) = X(X-1)(X-a)$.
Tracer sur $\left[-2, 1; 2, 1\right]$ la courbe de la fonction $d : t \mapsto \det(M_{P,Q_t})$.

Exercice 19 Centrale 2017-Python

Un plateau de jeu de type monopoly comporte un circuit de 12 cases numérotées de 0 à 11. Le joueur commence sur la case 0.

On note Y_n la variable aléatoire égale au numéro sur lequel se trouve le joueur après le n-ième lancer d'un dé classique équilibré à six faces. On suppose que les lancers sont mutuellement indépendants.

1. Justifier que $\forall n \in \mathbb{N}, Y_n(\Omega) \subset [\![0;11]\!]$, et donner la loi de Y_0.
2. Écrire une fonction de paramètre n et évaluant Y_n pour une réalisation aléatoire du jeu.
3. Afficher les fréquences, sur la réalisation aléatoire de 5000 parties du jeu, de l'événement $(Y_n(\omega) = k)$ pour $n \in \{50, 100, 200, 500\}$.
4. Exprimer $\mathrm{P}(Y_{n+1} = k)$ en fonction des $(\mathrm{P}(Y_n = i))_{0 \leq i \leq 11}$.
5. Soit $U_n = {}^t(\mathrm{P}(Y_n = 0), \mathrm{P}(Y_n = 1), \cdots, \mathrm{P}(Y_n = 11)) \in \mathcal{M}_{12,1}(\mathbb{R})$.
   Montrer qu'il existe une matrice P telle que $\forall n \in \mathbb{N},\ U_{n+1} = PU_n$.
6. Exprimer U_n en fonction de U_0. P est-elle diagonalisable ?

Exercice 20 Centrale 2017-Python

On pose $f_n(x) = \prod_{0 \leq i \leq n} \frac{1}{x+i}$, $F(x) = \sum_{n=0}^{+\infty} f_n(x)$ et $G(x) = \sum_{n=0}^{+\infty} \frac{(-1)^n}{n!(x+n)}$.

1. Montrer que F et G sont bien définies sur $\mathbb{R}_+^*$.
2. Tracer les courbes de F et G sur des intervalles raisonnables. Tracer ensuite $\frac{F}{G}$. Que peut-on conjecturer ?

3.1. Justifier l'existence d'une famille $(\ a_{i,n})$ telle que pour tout $n \in \mathbb{N}$, $f_n(x) = \sum_{i=0}^{n} \frac{a_{i,n}}{x+i}$.

À l'aide d'équivalents au voisinage des points $-i$, exprimer les coefficients $a_{i,n}$.

3.2. Montrer alors la conjecture faite à la question précédente.

Exercice 21 Centrale 2017-Python

Soit $f : x \mapsto \frac{1}{2-e^x}$, et on note $\forall n \in \mathbb{N},\ a(n) = \frac{f^{(n)}(0)}{n!}$.

1. Montrer que $\forall n \in \mathbb{N},\ a(n) = \sum_{k=1}^{n} \frac{a(n-k)}{k!}$. (On pourra utiliser la fonction $x \mapsto (2-e^x)f(x)$).
2. Écrire, avec Python, une procédure permettant de calculer $a(n)$ pour $n \in [\![0;10]\!]$.
3. Tracer les courbes $(n, a(n))$, $\left(n, \frac{1}{(\ln(2))^n}\right)$ et $\left(n, \frac{1}{2(\ln(2))^n}\right)$. En déduire le rayon de convergence de $\sum a(n)x^n$. On note S sa fonction somme.
4. Tracer l'approximation de la courbe de S grâce aux sommes partielles pour $n \in [\![0;6]\!]$.
5. Tracer la courbe de f sur $\big[0;10\big]$. que peut-on en déduire ? Le démontrer.

Exercice 22 Centrale 2017-Python

Un plateau de jeu de type monopoly comporte un circuit de 12 cases numérotées de 0 à 11. Le joueur commence sur la case 0.

On note Y_n la variable aléatoire égale au numéro sur lequel se trouve le joueur après le n-ième lancer d'un dé classique équilibré à six faces. On suppose que les lancers sont mutuellement indépendants.

1. Justifier que $\forall n \in \mathbb{N}, Y_n(\Omega) \subset [\![0;11]\!]$, et donner la loi de Y_0.
2. Écrire une fonction de paramètre n et évaluant Y_n pour une réalisation aléatoire du jeu.
3. Afficher les fréquences, sur la réalisation aléatoire de 5000 parties du jeu, de l'événement $(Y_n(\omega) = k)$ pour $n \in \{50, 100, 200, 500\}$.
4. Exprimer $\mathrm{P}(Y_{n+1} = k)$ en fonction des $(\mathrm{P}(Y_n = i))_{0 \leq i \leq 11}$.
5. Soit $U_n = {}^t(\mathrm{P}(Y_n = 0), \mathrm{P}(Y_n = 1), \cdots, \mathrm{P}(Y_n = 11)) \in \mathcal{M}_{12,1}(\mathbb{R})$.
   Montrer qu'il existe une matrice P telle que $\forall n \in \mathbb{N},\ U_{n+1} = PU_n$.
6. Exprimer U_n en fonction de U_0. P est-elle diagonalisable ?

Exercice 23 Centrale 2017-Python

On utilise une pièce dont le lancer donne pile avec la probabilité $p \in]0;1[$.

On note E_n l'événement : "ne pas obtenir 2 piles d'affilée au cours des n premiers lancers". Soit $p_n = \mathrm{P}(E_n)$.

1. Écrire un programme en python qui prend n et p pour paramètres et renvoie **True** si E_n est réalisé et **False** sinon.
2. Montrer que $\forall n \in \mathbb{N},\ p_{n+2} = (1-p)p_{n+1} + p(1-p)p_n$. et en déduire que l'événement "obtenir deux piles d'affilée sur un nombre infini de lancers" est presque sûr.
3. Écrire une fonction en python de paramètre n qui donne la probabilité p_n, pour p fixé.
4. Soit $n \in \mathbb{N}^*$ et T la variable aléatoire égale à n lorsque l'on obtient pile aux lancers $n-1$ et n, E_{n-1} étant réalisé.
   Écrire une fonction en python de paramètre d'entrée n donnant $\mathrm{P}(T = n)$.
5. Donner la fonction génératrice de T, montrer que T admet une espérance et la calculer.
6. Écrire un programme qui vérifie la valeur de cette espérance.

Exercice 24 ENSAM. python 2017

1. Soit $p \in \big[0;1\big]$. Écrire une fonction prenant en argument p et renvoyant la valeur 1 avec probabilité p, la valeur -1 avec probabilité $1-p$.
2. Soit $(X_n)_{n\in\mathbb{N}}$ une suite de variables aléatoires mutuellement indépendantes telles que :
   pour tout $n \in \mathbb{N}$, $\mathrm{P}(X_n = 1) = p$ et $\mathrm{P}(X_n = -1) = 1-p$.
   On pose $S_0 = 0$ et pour tout $n \in \mathbb{N}^*$, $S_n = X_1 + X_2 + \cdots + X_n$.
   Écrire une fonction d'arguments p et n, et qui renvoie la liste $\big[S_0, S_1, \cdots, S_n\big]$.
   Conjecturer la limite de la suite (S_n) en fonction de p.
3. Soit $n \in \mathbb{N}^*$. On note $T_n = \min\{k \in \mathbb{N},\ |S_k| = n\}$. Écrire une fonction d'arguments p, n et un entier $N > 0$, qui renvoie une liste de N tirages de la variable aléatoire T_n.

Exercice 25 ENSAM 2017 - Python

1. Écrire une fonction **sym** prenant en argument $M \in \mathcal{M}_n(\mathbb{R})$ et qui renvoie **True** si M est symétrique et **False** sinon.
2. Écrire une fonction **decomp** prenant en argument M et qui renvoie l'unique couple $(A, S) \in \mathcal{A}_n(\mathbb{R}) \times \mathcal{S}_n(\mathbb{R})$ tel que $M = A + S$.
3. Écrire une fonction **ortho** prenant en argument $M \in \mathcal{M}_n(\mathbb{R})$ et qui renvoie **True** si M est orthogonale et **False** sinon.

Exercice 26 Centrale 2017-Python
Soit g une fonction de classe $\mathcal{C}^1$ sur $\mathbb{R}_+^*$, vérifiant les conditions suivantes :
$\forall t \in \mathbb{R}_+^*,\ g(t+1) - g(t) = \arctan\left(\dfrac{1}{\sqrt{t}}\right)$; $g(1) = 0$ et $\lim\limits_{t\to+\infty} g'(t) = 0$.

1. Déterminer g' sous forme de somme d'une série de fonctions. *Ind.* Considérer $g'(t+1) - g'(t)$.
2. Déterminer g sous forme de somme d'une série de fonctions.
3. Soit $(z_n)_{n\in\mathbb{N}^*}$ la suite telle que $z_1 = 1$ et $\forall n \in \mathbb{N}^*,\ z_{n+1} = z_n + i\dfrac{z_n}{|z_n|}$.
   Montrer que cette suite est bien définie.
4. Calculer les dix premiers termes de la suite (z_n).
5. Tracer $z_1, z_2, \cdots z_{10}$ dans le plan complexe, et sur la même figure l'arc paramétré défini par
   $\begin{cases} x(t) &= \sqrt{t}\cos(g(t)) \\ y(t) &= \sqrt{t}\sin(g(t)) \end{cases}$
6. Formuler une conjecture puis la démontrer.
7. Trouver un équivalent de $g(n)$ lorsque $n \to +\infty$.

Exercice 27 ENSAM 2017 - Python
On veut tracer la courbe C d'équation $f(x, y) = 0$ si $f(x, y) = x^4 + 2y^2 + 2xy$.

1. Écrire une fonction **f** qui prend en argument une liste u=**array**$([x, y])$ qui renvoie la valeur $f(x, y)$.
2. Écrire une fonction **nablaf** qui prend en argument une liste u=**array**$([x, y])$ qui renvoie la valeur $\nabla f(x, y)$.
3. Écrire une fonction **points** qui prend en argument un nombre pas de type float, une fonction f et en entier n de type int et qui renvoie la la liste des points A_k approchant $\mathcal{C}$, telle que :
   $A_0 \in \mathcal{C}$, et $\forall k \in [\![0; n-1]\!],\ \text{Vect}A_kA_{k+1} = pas * \text{Vect}V_t$
   (avec $pas > 0$, $n \geq 2$, et VectV_t un vecteur unitaire tangent à $\mathcal{C}$ en A_k).
4. Tracer $\mathcal{C}$ en utilisant la fonction **points**.

Exercice 28 Centrale 2017-Python
Soit $M \in \mathcal{M}_n(\mathbb{R})$. On définit $\varphi_M :\ \mathbb{R}^n \to \mathbb{R}$ définie par $\varphi_M(X) = {}^tXMX$.

1. Exemple : dans cette question, $n = 2$ et $M = \begin{pmatrix} -2 & 4 \\ -1 & 2 \end{pmatrix}$.

1.1. Soit $X = \begin{pmatrix} x \\ y \end{pmatrix}$. Tracer la surface d'équation $z = \varphi_M(x, y)$ pour $(x, y) \in \big[-2; 2\big]^2$.

1.2. Déterminer les éventuels extrema locaux de φ_M.

1.3. Montrer que φ_M est surjective de $\mathbb{R}^2$ sur $\mathbb{R}$.

2. On revient au cas général.

2.1. Montrer que M peut se décomposer en somme d'une matrice symétrique S et d'une matrice antisymétrique A. Cette décomposition est-elle unique ?

2.2. Écrire en Python une fonction qui donne la partie symétrique d'une matrice carrée M.

2.3. Montrer que $\varphi_M = \varphi_S$.

3. On suppose que M est nilpotente.

3.1. Montrer que $\mathrm{tr}(M) = \mathrm{tr}(S)$.

3.2. Trouver le spectre de M.

3.3. Déterminer $\varphi_M(\mathbb{R}^n)$.

CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Analyse numérique

La plupart des fonctions présentées dans cette section nécessitent l'import du module `numpy` et de sous-modules du module `scipy`. Les instructions nécessaires aux exemples suivants sont listés ci-dessous.

```
import numpy as np
import scipy.optimize as resol
import scipy.integrate as integr
import matplotlib.pyplot as plt
```

Nombres complexes

Python calcule avec les nombres complexes. Le nombre imaginaire pur i se note `1j`. Les attributs `real` et `imag` permettent d'obtenir la partie réelle et la partie imaginaire. La fonction `abs` calcule le module d'un complexe.

```
>>> a = 2 + 3j
>>> b = 5 - 3j
>>> a*b
(19+9j)
>>> a.real
2.0
>>> a.imag
3.0
>>> abs(a)
3.6055512754639896
```

Fonctions mathématiques

La constante π s'obtient grâce à la commande `pi`.

Le module `numpy` connaît les fonctions mathématiques usuelles. La fonction partie entière s'obtient par la commande `floor`. Attention la fonction logarithme népérien a pour nom de commande `log`.

```
>>> np.exp(1)
2.7182818284590451
>>> np.cos(np.pi)
-1.0
>>> np.log(np.exp(1))
1.0
>>> np.floor(3.4)
3
>>> np.floor(-3.7)
-4
```

Résolution approchée d'équations

Pour résoudre une équation du type $f(x) = 0$ où f est une fonction d'une variable réelle, on peut utiliser la fonction `fsolve` du module `scipy.optimize`. Il faut préciser la valeur initiale x_0 de l'algorithme employé par la fonction `fsolve`. Le résultat peut dépendre de cette condition initiale.

```
def f(x):
    return x**2 - 2

>>> resol.fsolve(f, -2.)
array([-1.41421356])

>>> resol.fsolve(f, 2.)
array([ 1.41421356])
```

Dans le cas d'une fonction f à valeurs vectorielles, on utilise la fonction `root`. Par exemple, pour résoudre le système non linéaire

$$\begin{cases} x^2 - y^2 = 1 \\ x + 2y - 3 = 0 \end{cases}$$

```
def f(v):
    return v[0]**2 - v[1]**2 - 1, v[0] + 2*v[1] - 3

>>> sol = resol.root(f, [0,0])
>>> sol.success
True
>>> sol.x
array([ 1.30940108,  0.84529946])

>>> sol=resol.root(f, [-5,5])
>>> sol.success
True
>>> sol.x
array([-3.30940108,  3.15470054])
```

Calcul approché d'intégrales

La fonction `quad` du module `scipy.integrate` permet de calculer des valeurs approchées d'intégrales. Elle renvoie une valeur approchée de l'intégrale ainsi qu'un majorant de l'erreur commise. Cette fonction peut aussi s'employer avec des bornes d'intégration égales à $+\infty$ ou $-\infty$.

```
def f(x):
    return np.exp(-x)

>>> integr.quad(f, 0, 1)
(0.6321205588285578, 7.017947987503856e-15)

>>> integr.quad(f, 0, np.inf)
(1.0000000000000002, 5.842607038578007e-11)
```

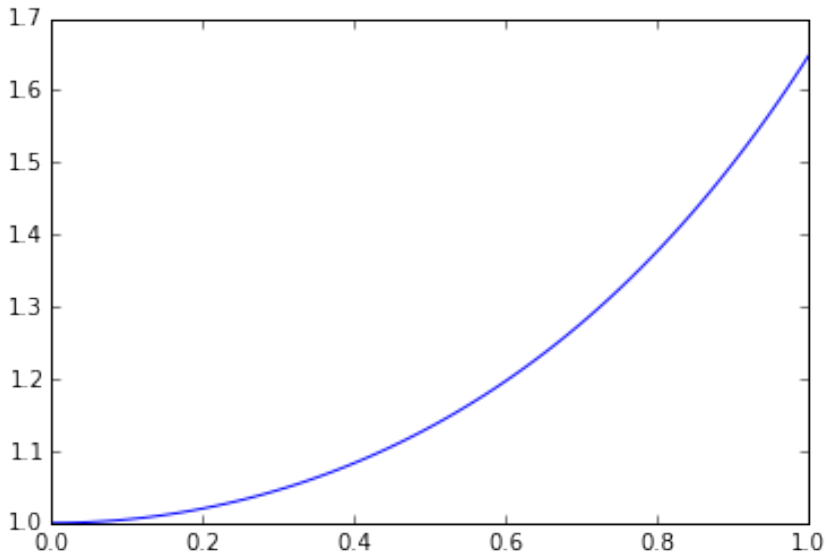
Cette fonction peut être employée pour la définition d'intégrales à paramètres. Ainsi si on veut obtenir des valeurs approchées de $\Gamma(x) = \int_0^{+\infty} e^{-t}t^{x-1}\mathrm{d}t$ pour x réel strictement positif on pourra procéder ainsi :

```
def g(x):
    def f(t):
        return np.exp(-t)*t**(x-1)
    return integr.quad(f,0,np.inf)[0]

>>> g(2)
0.9999999999999998
```

Résolution approchées d'équations différentielles

Pour résoudre une équation différentielle $x' = f(x,t)$, on peut utiliser la fonction `odeint` du module `scipy.integrate`. Cette fonction nécessite une liste de valeurs de t, commençant en t_0, et une condition initiale x_0. La fonction renvoie des valeurs approchées (aux points contenus dans la liste des valeurs de t) de la solution x de l'équation différentielle qui vérifie $x(t_0) = x_0$. Pour trouver des valeurs approchées sur $[0,1]$ de la solution $x'(t) = tx(t)$ qui vérifie $x(0) = 1$, on peut employer le code suivant.

```
def f(x, t):
    return t*x

>>> T = np.arange(0, 1.01, 0.01)
>>> X = integr.odeint(f, 1, T)
>>> X[0]
array([ 1.])
>>> X[-1]
array([ 1.64872143])
>>> plt.plot(T,X)
>>> plt.show()
```

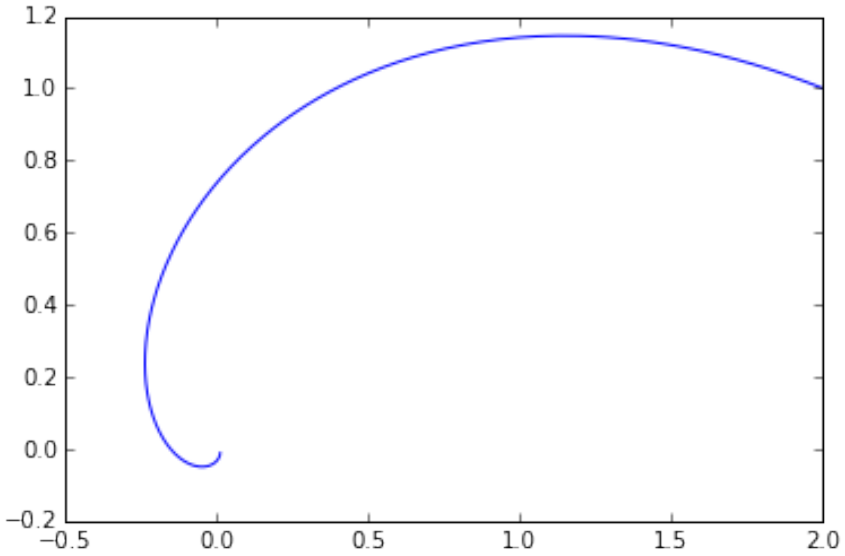


Si on veut résoudre, sur $[0,1]$, le système différentiel

$$\begin{cases} x'(t) = -x(t) - y(t) \\ y'(t) = x(t) - y(t) \end{cases}$$

avec la condition initiale $x(0) = 2, y(0) = 1$ le code devient le suivant.

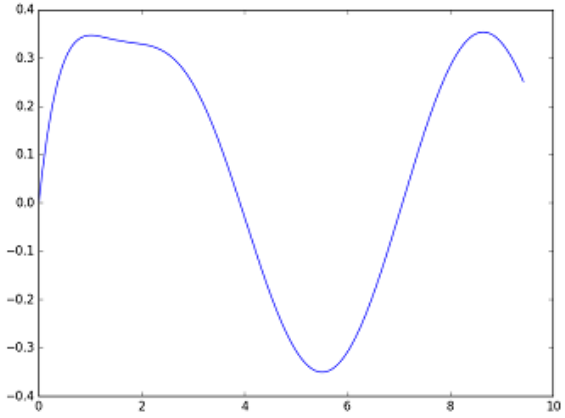
```
def f(x, t):
    return np.array([-x[0]-x[1], x[0]-x[1]])

>>> T = np.arange(0, 5.01, 0.01)
>>> X = integr.odeint(f, np.array([2.,1.]), T)
>>> X[0]
array([ 2.,  1.])
>>> plt.plot(X[:,0], X[:,1])
>>> plt.show()
```



Pour résoudre une équation différentielle scalaire d'ordre 2 de solution x, on demandera la résolution du système différentiel d'ordre 1 satisfait par $X(t) = \begin{pmatrix} x(t) \\ x'(t) \end{pmatrix}$.

Ainsi, si on considère la fonction x qui vérifie l'équation différentielle $x''(t) + 2x'(t) + 3x(t) = \sin(t)$ avec les conditions initiales $x(0) = 0$, $x'(0) = 1$ et , le vecteur X vérifiera $X'(t) = \begin{pmatrix} x'(t) \\ -2x'(t) - 3x(t) - \sin(t) \end{pmatrix}$. Pour obtenir la représentation graphique de x sur l'intervalle $[0, 3\pi]$, on pourra utiliser le code suivant :

```
def f(x,t):
    return np.array([x[1], -2*x[1] - 3*x[0] + np.sin(t)])

T = np.arange(0, 3*np.pi + 0.01, 0.01)
X = integr.odeint(f, np.array([0,1]), T)
plt.plot(T, X[:,0])
plt.show()
```



CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Calcul matriciel

On travaille avec les modules `numpy` et `numpy.linalg`.

```
import numpy as np
import numpy.linalg as alg
```

Création de matrices

Pour définir une matrice, on utilise la fonction `array` du module `numpy`.

```
>>> A = np.array([[1, 2, 3], [4, 5, 6]])
>>> A
array([[1, 2, 3],
       [4, 5, 6]])
```

L'attribut `shape` donne la taille d'une matrice : nombre de lignes, nombre de colonnes. On peut redimensionner une matrice, sans modifier ses termes, à l'aide de la méthode `reshape`.

```
>>> A.shape
(2, 3)
>>> A = A.reshape((3, 2))
>>> A
array([[1, 2],
       [3, 4],
       [5, 6]])
```

L'accès à un terme de la matrice `A` se fait à l'aide de l'opération d'indexage `A[i, j]` où `i` désigne la ligne et `j` la colonne. **Attention, les indices commencent à zéro !** À l'aide d'intervalles, on peut également récupérer une partie d'une matrice : ligne, colonne, sous-matrice. Rappel, `a:b` désigne l'intervalle ouvert à droite $[\![a, b[\![$, `:` désigne l'intervalle contenant tous les indices de la dimension considérée. Notez la différence entre l'indexation par un entier et par un intervalle réduit à un entier.

```
>>> A[1, 0]     # terme de la deuxième ligne, première colonne
3
>>> A[0, :]     # première ligne sous forme de tableau à 1 dimension
array([1, 2])
>>> A[0, :].shape
(2,)
>>> A[0:1, :]   # première ligne sous forme de matrice ligne
array([[1, 2]])
>>> A[0:1, :].shape
(1, 2)
>>> A[:, 1]     # deuxième colonne sous forme de tableau à 1 dimension
>>> array([2, 4, 6])
A[:, 1:2]   # deuxième colonne sous forme de matrice colonne
array([[2],
       [4],
       [6]])
>>> A[1:3, 0:2] # sous-matrice lignes 2 et 3, colonnes 1 et 2
array([[3, 4],
       [5, 6]])
```

Les fonctions `zeros` et `ones` permettent de créer des matrices remplies de 0 ou de 1. La fonction `eye` permet de créer une matrice du type I_n où n est un entier. La fonction `diag` permet de créer une matrice diagonale.

```
>>> np.zeros((2,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.ones((3,2))
array([[ 1.,  1.],
       [ 1.,  1.],
       [ 1.,  1.]])
>>> np.eye(4)
array([[ 1.,  0.,  0.,  0.],
       [ 0.,  1.,  0.,  0.],
       [ 0.,  0.,  1.,  0.],
       [ 0.,  0.,  0.,  1.]])
>>> np.diag([1,2,3])
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

Enfin la fonction `concatenate` permet de créer des matrices par blocs en superposant (`axis=0`) ou en plaçant côte à côte (`axis=1`) plusieurs matrices.

```
>>> A = np.ones((2,3))
>>> B = np.zeros((2,3))
>>> np.concatenate((A,B), axis=0)
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.concatenate((A,B), axis=1)
array([[ 1.,  1.,  1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  0.,  0.,  0.]])
```

Quelques méthodes ou fonctions utiles avec les tableaux Numpy

Pour copier un tableau, il est recommandé d'utiliser la méthode `copy`.

```
>>> A = np.array([[1,-2,3], [-4,5,-6]])
>>> B = A.copy()
>>> B[1, 0] = 8
>>> A
array([[ 1, -2,  3],
       [-4,  5, -6]])
>>> B
array([[ 1, -2,  3],
       [ 8,  5, -6]])
```

Les fonctions `amax`,`amin` et `mean` du module `numpy` permettent respectivement de calculer le maximum, le minimum et la moyenne des éléments d'un tableau.

```
>>> np.amax(A)
5
>>> np.amin(A)
-6
>>> np.mean(A)
-0.5
```

Enfin la commande `array_equal` permet de tester l'égalité terme à terme de deux tableaux de même taille.

```
>>> np.array_equal(A, B)
False
```

Calcul matriciel

Les opérations d'ajout et de multiplication par un scalaire se font avec les opérateurs `+` et `*`.

```
>>> A = np.array([[1,2], [3,4]])
>>> B = np.eye(2)
>>> A + 3*B
array([[ 4.,  2.],
       [ 3.,  7.]])
```

Pour effectuer un produit matriciel (lorsque que cela est possible), il faut employer la fonction `dot`.

```
>>> A = np.array([[1,2], [3,4]])
>>> B = np.array([[1,1,1], [2,2,2]])
>>> np.dot(A, B)
array([[ 5,  5,  5],
       [11, 11, 11]])
```

On peut également utiliser la méthode `dot` qui est plus pratique pour calculer un produit de plusieurs matrices. Enfin la fonction `matrix_power` du module `numpy.linalg` permet de calculer des puissances de matrices.

```
>>> A.dot(B)
array([[ 5,  5,  5],
       [11, 11, 11]])
>>> A.dot(B).dot(np.ones((3,2)))
array([[ 15.,  15.],
       [ 33.,  33.]])
>>> alg.matrix_power(A,3)
array([[ 37,  54],
       [ 81, 118]])
```

La transposée s'obtient avec la fonction `transpose`. L'expression `A.T` renvoie aussi la transposée de `A`.

```
>>> np.transpose(B)
array([[1, 2],
       [1, 2],
       [1, 2]])
>>> B.T
array([[1, 2],
       [1, 2],
       [1, 2]])
```

Le déterminant, le rang et la trace d'une matrice s'obtiennent par les fonctions `det`, `matrix_rank` du module `numpy.linalg` et `trace` du module `numpy`. Enfin la fonction `inv` du module `numpy.linalg` renvoie l'inverse de la matrice s'il existe.

```
>>> alg.det(A)
-2.0000000000000004
>>> alg.matrix_rank(A)
2
>>> np.trace(A)
5
>>> alg.inv(A)
matrix([[-2. ,  1. ],
        [ 1.5, -0.5]])
```

Pour résoudre le système linéaire $Ax = b$ lorsque la matrice A est inversible, on peut employer la fonction `solve` du module `numpy.linalg`.

```
>>> b = np.array([1,5])
>>> alg.solve(A, b)
array([ 3., -1.])
```

Éléments propres d'une matrice

La fonction `poly` du module `numpy` appliquée à une matrice carrée renvoie la liste des coefficients du polynôme caractéristique par degré décroissant.

```
>>> A = np.array([[2,-4],[1,-3]])
>>> np.poly(A)
array([ 1.,  1., -2.])
```

La fonction `eigvals` du module `numpy.linalg` renvoie les valeurs propres de la matrice.

```
>>> alg.eigvals(A)
array([ 1., -2.])
```

Pour obtenir en plus les vecteurs propres associés, il faut employer la fonction `eig`. Cette fonction renvoie un tuple constitué de la liste des valeurs propres et d'une matrice carrée. La $i^{\text{ième}}$ colonne de cette matrice est un vecteur propre associé à la $i^{\text{ième}}$ valeur de la liste des valeurs propres. Dans l'exemple ci dessous, on peut conclure que $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ est un vecteur propre de A associé à la valeur propre -2. On vérifie aussi que A est diagonalisable.

```
>>> L = alg.eig(A)
>>> L
(array([ 1., -2.]), array([[ 0.9701425,  0.70710678],
       [ 0.24253563,  0.70710678]]))
>>> L[1][:,1]
array([ 0.70710678,  0.70710678])
>>> L[1].dot(np.diag(L[0])).dot(alg.inv(L[1]))
array([[ 2., -4.],
       [ 1., -3.]])
```

Produit scalaire et produit vectoriel

La fonction `vdot` permet de calculer le produit scalaire de deux vecteurs de $\mathbb{R}^n$.

```
>>> u = np.array([1,2])
>>> v = np.array([3,4])
>>> np.vdot(u, v)
11
```

La fonction `cross` permet de calculer le produit vectoriel de deux vecteurs de $\mathbb{R}^3$.

```
>>> u = np.array([1,0,0])
>>> v = np.array([0,1,0])
>>> np.cross(u, v)
array([0, 0, 1])
```

CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Réalisation de tracés

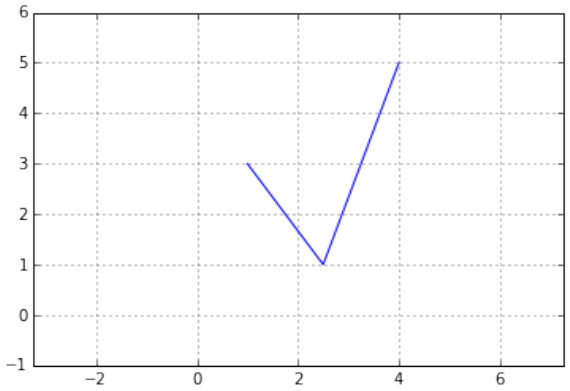
Les fonctions présentées dans ce document permettent la réalisation de tracés. Elles nécessitent l'import du module `numpy` et du module `matplotlib.pyplot`. De plus pour effectuer des tracés en dimension 3, il convient d'importer la fonction `Axes3d` du module `mpl_toolkits.mplot3d`. Les instructions nécessaires aux exemples qui suivent sont listés ci-dessous.

```
import math
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
```

Tracés de lignes brisées et options de tracés

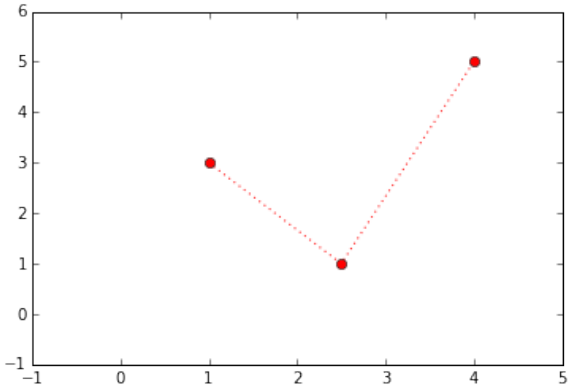
On donne la liste des abscisses et la liste des ordonnées puis on effectue le tracé. La fonction `axis` permet de définir la fenêtre dans laquelle est contenue le graphique. L'option `equal` permet d'obtenir les mêmes échelles sur les deux axes. Les tracés relatifs à divers emplois de la fonction `plot` se superposent. La fonction `plt.clf()` efface les tracés contenus dans la fenêtre graphique.

```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis('equal')
plt.plot(x, y)
plt.axis([-1., 5., -1., 6.])
plt.grid()
plt.show()
```



La fonction `plot` admet de nombreuses options de présentation. Le paramètre `color` permet de choisir la couleur (`'g'` : vert, `'r'` : rouge, `'b'` : bleu). Pour définir le style de la ligne, on utilise `linestyle` (`'-'` : ligne continue, `'--'` : ligne discontinue, `':'` : ligne pointillée). Si on veut marquer les points des listes, on utilise le paramètre `marker` (`'+'`, `'.'`, `'o'`, `'v'` donnent différents symboles).

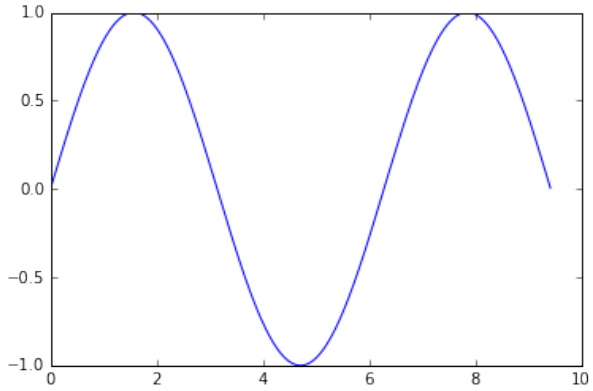
```
x = [1., 2.5, 4.]
y = [3., 1., 5.]
plt.axis([-1., 5., -1., 6.])
plt.plot(x, y, color='r', linestyle=':',
         marker='o')
plt.show()
```



Tracés de fonction

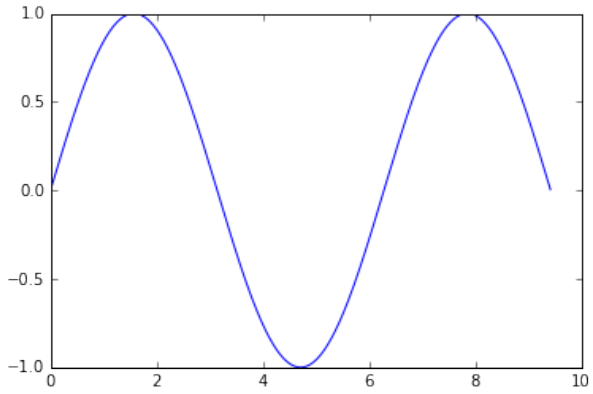
On définit une liste d'abscisses puis on construit la liste des ordonnées correspondantes. L'exemple ci-dessous trace $x \mapsto \sin x$ sur $[0, 3\pi]$.

```
def f(x):
    return math.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = [ f(x) for x in X ]
plt.plot(X, Y)
plt.show()
```



Il est généralement plus intéressant d'utiliser les fonctions du module `numpy`, plutôt que celles du module `math`, car elles permettent de travailler aussi bien avec des scalaires qu'avec des tableaux (on les appelle fonctions vectorisées et *universal function* ou `ufunc` dans la documentation officielle de Python).

```
def f(x):
    return np.sin(x)

X = np.arange(0, 3*np.pi, 0.01)
Y = f(X)
plt.plot(X, Y)
plt.show()
```



Une autre solution consiste à utiliser la fonction `vectorize` du module `numpy` qui permet de transformer une fonction scalaire en une fonction capable de travailler avec des tableaux. Il est cependant beaucoup plus efficace d'utiliser directement des fonctions vectorisées.

```
def f(x):
    return math.sin(x)

f = np.vectorize(f)
```

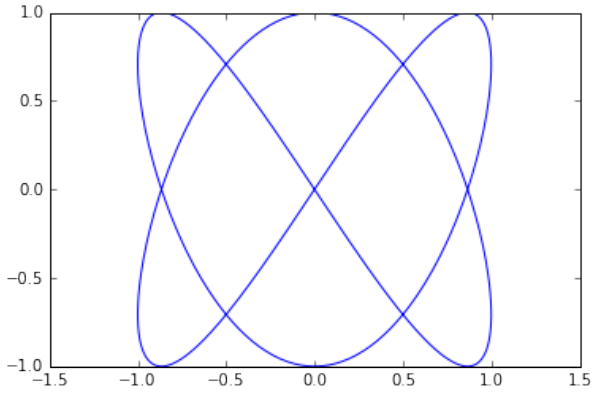
Il est à noter que les opérateurs python (`+`, `-`, `*`, etc.) peuvent s'appliquer à des tableaux, ils agissent alors terme à terme. Ainsi la fonction `f` définie ci-dessous est une fonction vectorisée, elle peut travailler aussi bien avec deux scalaires qu'avec deux tableaux et même avec un scalaire et un tableau.

```
def f(x, y):
    return np.sqrt(x**2 + y**2)

>>> f(3, 4)
5.0
>>> f(np.array([1, 2, 3]), np.array([4, 5, 6]))
array([ 4.12310563,  5.38516481,  6.70820393])
>>> f(np.array([1, 2, 3]), 4)
array([ 4.12310563,  4.47213595,  5.        ])
```

Tracés d'arcs paramétrés

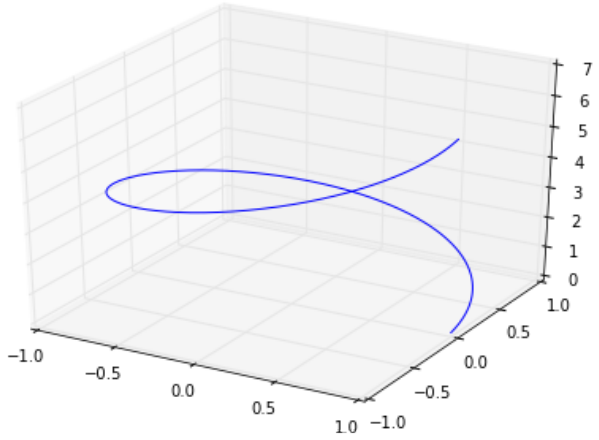
Dans le cas d'un arc paramétré plan, on définit d'abord la liste des valeurs données au paramètre puis on construit la liste des abscisses et des ordonnées correspondantes. On effectue ensuite le tracé.

```
def x(t):
    return np.sin(2*t)

def y(t):
    return np.sin(3*t)

T = np.arange(0, 2*np.pi, 0.01)
X = x(T)
Y = y(T)
plt.axis('equal')
plt.plot(X, Y)
plt.show()
```



Voici un exemple de tracé d'un arc paramétré de l'espace.

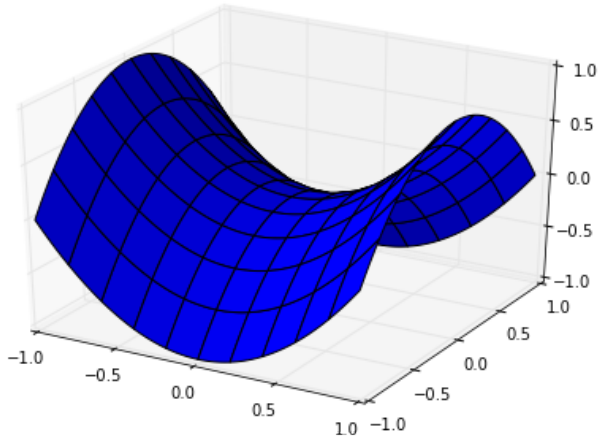
```
ax = Axes3D(plt.figure())
T = np.arange(0, 2*np.pi, 0.01)
X = np.cos(T)
Y = np.sin(T)
Z = T
ax.plot(X, Y, T)
plt.show()
```



Tracé de surfaces

Pour tracer une surface d'équation $z = f(x, y)$, on réalise d'abord une grille en (x, y) puis on calcule les valeurs de z correspondant aux points de cette grille. On fait ensuite le tracé avec la fonction `plot_surface`.

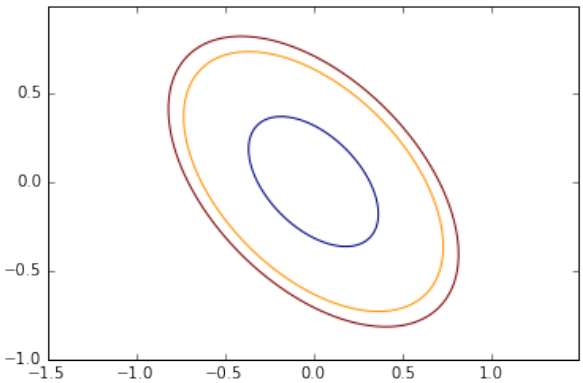
```
ax = Axes3D(plt.figure())

def f(x,y):
    return x**2 - y**2

f=np.vectorize(f)
X = np.arange(-1, 1, 0.02)
Y = np.arange(-1, 1, 0.02)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
ax.plot_surface(X, Y, Z)
plt.show()
```



Tracé de lignes de niveau

Pour tracer des courbes d'équation $f(x,y)=k$, on fait une grille en x et en y sur laquelle on calcule les valeurs de f. On emploie ensuite la fonction `contour` en mettant dans une liste les valeurs de k pour lesquelles on veut tracer la courbe d'équation $f(x,y)=k$.

```
def f(x,y):
    return x**2 + y**2 + x*y

f=np.vectorize(f)
X = np.arange(-1, 1, 0.01)
Y = np.arange(-1, 1, 0.01)
X, Y = np.meshgrid(X, Y)
Z = f(X, Y)
plt.axis('equal')
plt.contour(X, Y, Z, [0.1,0.4,0.5])
plt.show()
```



CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Polynômes

La classe `Polynomial` du module `numpy.polynomial.polynomial` permet de travailler avec des polynômes.

```
from numpy.polynomial import Polynomial
```

Pur créer un polynôme, il faut lister ses coefficients par ordre de degré croissant. Par exemple, pour le polynôme $X^3 + 2X - 3$,

```
p = Polynomial([-3, 2, 0, 1])
```

On peut alors utiliser cette variable comme une fonction pour calculer, en un point quelconque, la valeur de la fonction polynôme associée. Cette fonction peut agir également sur un tableau de valeurs, elle calcule alors la valeur de la fonction polynôme en chacun des points indiqués.

```
>>> p(0)
-3.0
>>> p([1, 2, 3])
array([  0.,   9.,  30.])
```

L'attribut `coef` donne accès aux coefficients ordonnés par degré croissant ; ainsi `p.coef[i]` correspond au coefficient du terme de degré `i`. La méthode `degree` renvoie le degré du polynôme alors que `roots` calcule ses racines.

```
>>> p.coef
array([-3.,  2.,  0.,  1.])
>>> p.coef[1]
2.0
>>> p.degree()
3
>>> p.roots()
array([-0.5-1.6583124j, -0.5+1.6583124j,  1.0+0.j       ])
```

La méthode `deriv` renvoie un nouveau polynôme, dérivé du polynôme initial. Cette méthode prend en argument facultatif un entier positif indiquant le nombre de dérivations à effectuer. De la même manière la méthode `integ` intègre le polynôme, elle prend un paramètre optionnel supplémentaire donnant la constante d'intégration à utiliser, ce paramètres peut être une liste en cas d'intégration multiple ; les constantes d'intégration non précisées sont prises égales à zéro.

```
>>> p.deriv().coef
array([ 2.,  0.,  3.])
>>> p.deriv(2).coef
array([ 0.,  6.])
>>> p.deriv(5).coef
array([-0.])
>>> p.integ().coef
array([ 0.  , -3.  ,  1.  ,  0.  ,  0.25])
>>> p.integ(1, 2).coef  # intégrer une fois avec la constante 2
array([ 2.  , -3.  ,  1.  ,  0.  ,  0.25])
>>> p.integ(2, [1, 2]).coef  # intégrer deux fois
array([ 2.        ,  1.        , -1.5       ,  0.33333333,  0.        ,
        0.05      ])
```

Les opérateurs `+`, `-`, `*` permettent d'additionner, soustraire et multiplier des polynômes. Ils fonctionnent également entre un polynôme et un scalaire. L'opérateur `**` permet d'élever un polynôme à une puissance entière positive. Enfin, on peut composer deux polynômes (`p(q)` remplace l'indéterminée X par le polynôme `q` dans le polynôme `p`)

```
>>> a = Polynomial([1, 2, 1])
>>> b = Polynomial([5, 3])
>>> p = 2*a * b + Polynomial([-7, 2])
>>> p.coef
array([  3.,  28.,  22.,   6.])
>>> (p**2).coef
array([    9.,   168.,   916.,  1268.,   820.,   264.,    36.])
>>> a(b).coef
array([ 36., 36., 9.])
```

L'opérateur `/` permet de diviser un polynôme par un scalaire. Pour diviser deux polynômes il faut utiliser l'opérateur `//` qui renvoie le quotient ; l'opérateur `%` calcule le reste.

```
>>> (p / 2).coef
array([  1.5,  14. ,  11. ,   3. ])
>>> q = p // a
>>> r = p % a
>>> q.coef
array([ 10.,   6.])
>>> r.coef
array([-7.,  2.])
>>> (q * a + r).coef
array([  3.,  28.,  22.,   6.])
```

CONCOURS CENTRALE•SUPÉLEC

Oral

Python

MP, PC, PSI, TSI

Probabilités

Les fonctions d'échantillonnage et de génération de valeurs pseudo-aléatoires sont regroupées dans la bibliothèque `numpy.random`.

```
import numpy.random as rd
```

L'expression `randint(a, b)` permet de choisir un entier au hasard dans l'intervalle $[\![a, b[\![$. La fonction `randint` prend un troisième paramètre optionnel permettant d'effectuer plusieurs tirages et de renvoyer les résultat sous forme de tableau ou de matrice.

```
>>> rd.randint(1, 7)          # un lancer de dé
2
>>> rd.randint(1, 7, 20)      # 20 lancers de dé
array([5, 2, 2, 3, 1, 5, 5, 3, 6, 4, 2, 6, 6, 4, 3, 2, 4, 5, 1, 3])
>>> rd.randint(1, 7, (4, 5))  # 20 lancers de dé sous forme d'une matrice 4x5
array([[3, 6, 1, 6, 3],
       [5, 1, 6, 2, 2],
       [3, 1, 2, 2, 5],
       [5, 2, 6, 1, 4]])
```

La fonction `random` renvoie un réel compris dans l'intervalle $[0, 1[$. Si X désigne la variable aléatoire correspondant au résultat de la fonction `random`, alors pour tout a et b dans $[0, 1]$ avec $a \leqslant b$, on a $P(a \leqslant X < b) = b - a$. Cette fonction accepte un paramètres optionnel permettant de réaliser plusieurs tirages et de les renvoyer sous forme de tableau ou de matrice.

```
>>> rd.random()
0.9168092013708049
>>> rd.random(4)
array([ 0.98748897,  0.86589972,  0.53683001,  0.50687386])
>>> rd.random((2,4))
array([[ 0.78230688,  0.83803526,  0.62077457,  0.27432819],
       [ 0.66522387,  0.71258365,  0.25813448,  0.28833084]])
```

La fonction `binomial` permet de simuler une variable aléatoire suivant une loi binomiale de paramètres `n` et `p`. Elle permet donc également de simuler une variable aléatoire suivant une loi de Bernoulli de paramètres `p` en prenant simplement $n = 1$. Cette fonction prend un troisième paramètre optionnel qui correspond, comme pour les fonctions précédentes, au nombre de valeurs à obtenir.

```
>>> rd.binomial(10, 0.3, 7)
array([2, 2, 2, 2, 2, 4, 3])
>>> rd.binomial(1, 0.6, 20)
array([0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1])
```

Les fonctions `geometric` et `poisson` fonctionnement de la même manière pour les lois géométrique ou de Poisson.

```
>>> rd.geometric(0.5, 8)
array([1, 1, 3, 1, 3, 2, 5, 1])
>>> rd.poisson(4, 15)
array([5, 2, 3, 4, 6, 0, 5, 3, 1, 5, 1, 5, 9, 4, 6])
```