Cahier de TP d’informatique

2025-2026

Table des matières

1 Nombres complexes **1**

I Représentation des nombres complexes 1

II Opérations algébriques sur les complexes 1

III Représentation géométrique des nombres complexes 2

IV Nombres complexes de module 1 2

V Équation $z^n = 1$ 3

2 Résolution approchée de l'équation $f(x) = 0$ **5**

I Données et hypothèses générales 5

I – A Rappel et principe de la méthode par balayage 5

I – B Rappel et prinicipe de la méthode par dichotomie 6

I – C Rappel et principe de la méthode de Newton 6

3 Notion de fonctions négligeables, équivalentes **9**

I Préliminaires mathématiques 9

II Croissances comparées, équivalents 9

II – A Échelle en $+\infty$ 9

II – B Équivalents 10

4 Intégration numérique **13**

I Principe de l'intégration numérique 13

II Méthode des rectangles 13

5 Méthode d'Euler et équations différentielles **17**

I Interprétation géométrique d'une équation différentielle 17

II Équation différentielle scalaire d'ordre 1 18

II – A Principe succintement rappelé 18

II – B Étude théorique d'un exemple historique 19

II – C Étude d'un second exemple 19

II – D Un exemple d'équation non autonome 20

III Système d'équations différentielles scalaires 20

III – A Contexte 20

IV Oscillateur harmonique non amorti 21

IV – A Présentation du modèle 21

IV – B Schéma d'Euler explicite 21

IV – C Schéma d'Euler implicite 22
IV – D Schéma d'Euler trapézoïdal 24

6 Suites réelles **25**
I Étude d'une suite implicite 25
II Étude d'une suite récurrente 25
II – A Fonctions de base 25
II – B Représentation graphique 26
II – C Obtention du diagramme de bifurcation 26

7 Polynômes **29**
I Choix de la structure de données 29
II Opérations sur les polynômes 30
II – A Multiplication par un scalaire 30
II – B Somme 30
II – C Produit 30
III Autour de la méthode de Hörner 30
III – A Présentation et programmation 30
III – B Factorisation d'un polynôme par $X - \alpha$ 31

8 Récursivité **33**
I Récursivité 33
I – A Présentation du concept 33
I – B Ce qu'il faut repérer pour programmer récursivement 33
II Exercices 34
II – A Récursivité sur des calculs de flottants 34
II – B Programmation récursive sur les objets structurés 35

9 Module `numpy` **37**
I Opérations de base 37
I – A Définition de matrices 37
I – B Construction de matrices, matrices utiles 38
II Produit matriciel 39
II – A Programmation 39
II – B Puissances successives 39
III Exercices 39
III – A Test d'appartenance 39
III – B Changement de base 39

10 Méthodes de tri **41**
I Modification d'un objet structuré par effet de bord 41
II Méthodes de secours 41
II – A Tri artisanal 41
II – B Tri artisanal bis 41

III Tri à bulles (bubblesort) . . . 42
IV Tri insertion . . . 42
V Tri rapide (quicksort) . . . 43
VI Applications . . . 43
VI – A Médiane d'une statistique univariée . . . 43
VI – B Tri d'une liste en fonction d'une autre . . . 44

11 Diagonalisation des matrices carrées 45
I Préliminaires . . . 45
II Diagonlisation . . . 45

12 Simulation de variables aléatoires discrètes 47
I Simulation d'expériences aléatoires . . . 47
I – A Simulation d'expériences aléatoires . . . 47
I – B Variable aléatoire de Bernoulli . . . 49
I – C Variable binomiale . . . 49
I – D Variable géométrique . . . 49
I – E Simulation d'une VAR discrète quelconque . . . 49

13 Simulation de variables aléatoires à densité 51
I Lois usuelles . . . 51
II Exercices . . . 52

14 Statistiques 53
I Autour de la loi normale centrée réduite . . . 53
II Autour du théorème central-limite . . . 53
II – A Quelques définitions . . . 53
II – B Théorème de Moivre-Laplace . . . 54
III Convergence en loi . . . 54
III – A Expérimentations . . . 54

15 Graphes 55
I Notion de graphe . . . 55
I – A Définition . . . 55
I – B Représentation graphique d'un graphe . . . 55
I – C Représentation matricielle d'un graphe . . . 56
I – D Implémentations possibles sous Python . . . 57
I – E Chemin dans un graphe . . . 57
II Parcours en largeur (algorithme BFS) . . . 58
II – A Principe et implémentation . . . 58
II – B Application . . . 59

Thème 1 : Nombres complexes

Avertissement : on n'utilise pas la représentation des nombres complexes prédéfinie sur python dans ce TP

I Représentation des nombres complexes

```
1 from math import pi
2 from math import cos,sin
3 import matplotlib.pyplot as plt
```

Choix de la structure de données : on décide de *représenter* en machine le nombre complexe $z = a + ib$ (a, b réels) par le *tuple de flottants* `(a,b)`.

[Q1.] Rappeler la différence essentielle en termes d'opérations entre liste et tuple sous python.

[Q2.] Écrire une fonction :

a) `rpart(z)` qui prend en entrée un [tuple qui représente un] complexe z et qui renvoie en sortie le flottant égal à sa partie réelle.

b) `ipart(z)` qui en fait autant avec la partie imaginaire.

II Opérations algébriques sur les complexes

[Q3.] **a)** Écrire une fonction `somme(z1,z2)` qui prend en entrée deux nombres complexes z_1, z_2 et qui renvoie en sortie le tuple représentant le complexe $z = z_1 + z_2$

b) Écrire une fonction `produit(z1,z2)` qui prend en entrée deux nombres complexes z_1, z_2 et qui renvoie en sortie le tuple représentant le complexe $z = z_1 \times z_2$.

c) Écrire une fonction `bar(z)` qui prend en entrée un nombre complexe z et qui renvoie en sortie le tuple représentant le complexe $\bar{z}$.

[Q4.] **a)** Écrire une fonction `module2(z)` qui prend en entrée un complexe z et qui renvoie en sortie le *flottant* $|z|^2$.

b) La relation $z\bar{z} = |z|^2$ est fausse en python. Pourquoi ?

c) Modifier la fonction de sorte qu'elle renvoie $|z|^2$ en tant que nombre complexe, et non pas en tant que flottant. (*on peut redéfinir à la suite de la première une fonction* `module2`.)

[Q5.] Écrire une fonction `inverse(z)` qui prend en entrée un nombre complexe z (non nul) et qui renvoie en sortie le tuple représentant le complexe $1/z$.

III Représentation géométrique des nombres complexes

[Q6.] Recopier les scirpts ci-dessous, qui serviront à tracer, pour un nombre complexe z donné, le point du plan d'affixe M et le vecteur $\overrightarrow{OM}$ du plan.

```
def texte(z):
    """ Entrée : z (tuple de floats). Un nombre complexe.
        Sortie : (str) la chaine de caractères représentant
                 z sous  forme algébrique.
    """
    a,b = z
    s = b>=0
    signe = '+'*s + '-'*(1-s)
    return str(a)+signe+str(abs(b))+'i'

def dessine(z,text=''):
    """ Entrée : z (tuple de floats). Un nombre complexe.
        Sortie : aucune. Trace le  point M d'affixe z,
                 et le vecteur OM.
    """
    a,b = z
    plt.plot([0,a],[0,b])
    plt.annotate(text,z)
    plt.arrow(0,0,a,b,width=0.05,length_includes_head=True)
```

[Q7.] Tester les commandes ci-dessous et comparer :

```
z = (1,-3)
dessine(z)
plt.grid('on')
# repère orthonormé
plt.gca().set_aspect('equal')
plt.show()
```

```
z=(1,-3)
dessine(z,texte(z))
plt.grid('on')
plt.gca().set_aspect('equal')
plt.show()
```

[Q8.] Choisir deux nombres complexes de son choix z_1, z_2 (ni réels, ni imaginaires purs !), calculer à la main le nombre complexe $S = z_1 + z_2$. Tracer graphiquement les points d'affixe z_1, z_2, S. Répéter avec d'autres valeurs de z_1 et z_2 jusqu'à avoir compris graphiquement l'effet de la somme de complexes.

IV Nombres complexes de module 1

[Q9.] Écrire une fonction `expi(theta)` qui prend en entrée un flottant θ et qui renvoie en sortie le [tuple représentant le nombre] complexe $e^{i\theta}$ (passer par la forme trigonométrique).

[Q10.] Choisir des valeurs de θ de votre choix, et tracer pour ces valeurs de θ, à l'aide de la fonction `dessine`, les points d'affixe z et $1/z$, où $z = e^{i\theta}$. Que constatez-vous ?

On fixe un nombre complexe C de votre choix, ainsi qu'un nombre réel θ, et on considère la suite de nombres complexes $(u_n)_{n\in\mathbb{N}}$ définie par récurrence de la manière suivante :

$$\begin{cases} & u_0 & = & C \\ \forall n \in \mathbf{N} & u_{n+1} & = & e^{i\theta} \times u_n \end{cases}$$

[Q11.] Écrire une fonction `u(C,theta,n)` qui prend en entrée un complexe C, un flottant θ, un entier n et qui renvoie en sortie la liste $[u_0, \ldots, u_n]$ des $(n+1)$ premiers termes de la suite (u_n). Attention : chaque terme de la suite est, du point de vue informatique, un tuple.

[Q12.] Donner la forme explicite de la suite (u_n).

[Q13.] Tracer les 15 premiers termes de la suite ainsi programmée. Interpréter géométriquement la multiplication d'un nombre complexe par $e^{i\theta}$. Pour y voir clair, un jeu de constantes pertinent est par exemple :

```
C=(1,0)
tour = 2*pi
N = 5
theta =  tour/N
n = N-1
```

V Équation $z^n = 1$

Soit $n \geq 2$ un entier fixé. On s'intéresse à l'équation d'inconnue $z \in \mathbf{C}$:

$$z^n = 1 \qquad (E)$$

[Q14.] Écrire une fonction `puissance(C,n)` qui prend en entrée un complexe C, un entier n, et qui renvoie en sortie le complexe C^n.

[Q15.] Tracer la suite des 10 premières puissances d'un nombre complexe C de votre choix. On choisira des exemples couvrant les deux cas suivants : $|C| > 1$ et $|C| < 1$. Que conjecturer sur le module d'une solution de (E) ?

[Q16.] À l'aide des questions précédentes, trouver une solution particulière (distincte de 1 !) à l'équation (E).

[Q17.] Montrer (mathématiquement) que les solutions de (E) sont les nombres $1 = \omega^0$, ω, ω^2, …, ω^{n-1} où $\omega = \exp(2i\pi/n)$. *Indication : on pourra prouver que toute solution est non nulle, puis chercher ces dernières sous forme exponentielle.*

[Q18.] Créer en python la liste $L_n = [\omega^0, \ldots, \omega^{n-1}]$:

a) À l'aide d'une boucle.

b) En compréhension.

On souhaite calculer les nombres $S_n = \sum_{k=0}^{n-1} \omega^k$ et $P_n = \prod_{k=0}^{n-1} \omega^k$

[Q19.] **a)** Créer une fonction en python qui prend en entrée un entier n et qui renvoie en sortie la liste des deux nombres complexes S_n et P_n.

b) La tester pour différentes valeurs de $n \geq 2$ et formuler une conjecture.

c) Calculer mathématiquement S_n et P_n.

Thème 2 : Résolution approchée de l'équation $f(x) = 0$

```python
import numpy as np
import matplotlib.pyplot as plt
```

I Données et hypothèses générales

a) (Séparation) On connaît un segment $[a, b]$ ne contenant qu'une seule racine $x_\star$ de f. Remarque : dans les cas généraux, l'obtention de la séparation demande une vraie étude dont on ne peut faire l'économie.

b) f est continue sur le segment $[a, b]$.

On cherche à calculer une valeur approchée de $x_\star$ à une constante $\varepsilon > 0$ donnée près, appelée précision.

Dans toute la suite, on sera amené à considérer l'équation (E) suivante sur un certain intervalle $[a, b]$:

$$(E) \qquad \cos x - x = 0 \quad \text{sur } [a, b].$$

[Q1.] Proposer graphiquement un intervalle $[a, b]$ contenant la solution $x_\star$ de (E).

I – A Rappel et principe de la méthode par balayage

a) *Principe :* Partant du point a, on évalue par boucle les valeurs de f aux points $x_k = a + k\varepsilon$, $k \in \mathbf{N}$.

b) *Critère d'arrêt :* dès que l'on a un changement de signe de $f(x_i)$ pour deux valeurs consécutives de x_k et x_{k+1}, on en déduit que $x_\star \in [x_k, x_{k+1}]$ (pourquoi ?). Comme $x_{k+1} - x_k = \varepsilon$, tout point de cet intervalle fournit une valeur approchée de $x_\star$ au moins à la précision ε.

[Q2.] **a)** Écrire une fonction `balayage(f,a,b,eps)` prenant en entrée une fonction `f`, des flottants `a`,`b`, et `eps` et renvoyant en sortie la solution de l'équation $f(x) = 0$ à la précision `eps` calculée par balayage, ainsi que le nombre `n` d'itérations effectuées pour l'obtenir.

b) Donner le résultat avec `eps`$= 10^{-7}$ pour l'équation (E).

I – B Rappel et prinicipe de la méthode par dichotomie

a) *Principe :* Partant du premier segment $[a,b]$, on construit, à chaque passage dans la boucle, un nouveau segment $[a_n,b_n]$ de longueur deux fois plus petite que celle du précédent, contenant la solution localisée $x_\star$ de l'équation et dont une des extrémités est soit a_{n-1}, soit b_{n-1}, l'autre étant par construction $c_n=(a_{n-1}+b_{n-1})/2$.

b) *Critère d'arrêt :* dès que la longueur du segment $[a_n,b_n]$ est de longueur $<\varepsilon$, tout point de cet intervalle fournit une valeur approchée de $x_\star$ au moins à la précision ε.

[Q3.] **a)** Écrire une fonction `dicho(f,a,b,eps)` prenant en entrée une fonction `f`, des flottants `a`, `b`, et `eps`, et renvoyant en sortie la solution de l'équation $f(x)=0$ à la précision `eps` calculée par dichotomie, ainsi que le nombre `n` d'itérations effectuées pour l'obtenir.

b) Donner le résultat avec `eps`$=10^{-7}$ pour l'équation (E).

c) Comparer le nombre d'itérations avec celui obtenu par la méthode par balayage.

I – C Rappel et principe de la méthode de Newton

a) f est $\mathscr{C}^2([a,b])$.

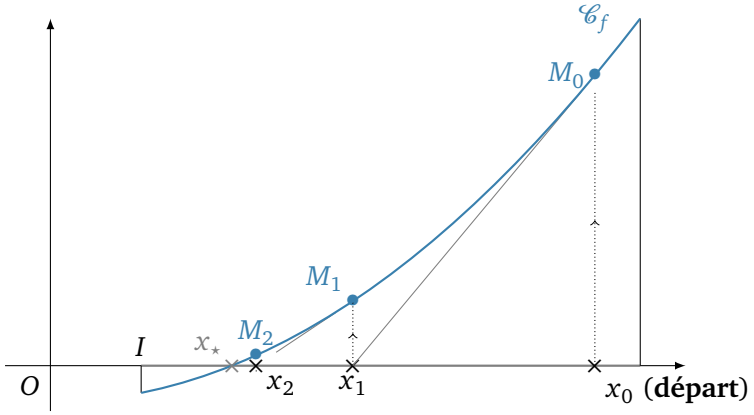
b) f' ne s'annule pas sur $[a,b]$ (f est donc strictement monotone sur I).

Principe : Partant d'un point $x_0\in[a,b]$, on considère $M_0(x_0,f(x_0))\in\mathscr{C}_f$, puis, de proche en proche, on construit des points M_n d'abscisse x_n tels que

$$\forall n\in\mathbf{N}\quad x_{n+1}=x_n-\frac{f(x_n)}{f'(x_n)}.$$

[Q4.] Mettons que l'on ait calculé le terme de rang k, x_k, de la suite de Newton $(x_n)_{n\geq 0}$. Rappeler l'équation de la tangente à $\mathscr{C}_f$ au point d'abscisse x_k.

Ainsi x_{k+1} s'obtient comme l'abscisse du point d'intersection de la tangente à $\mathscr{C}_f$ au point $M_k(x_k,f(x_k))$ avec l'axe des abscisses.



Critère d'arrêt : on adopte comme critère d'arrêt le suivant : si deux termes consécutifs x_n et x_{n+1} de la suite diffèrent de moins de ε, alors on considère que x_{n+1} est une valeur approchée de $x_\star$ à ε près.

[Q5.] **a)** Écrire une fonction prenant en entrée les flottants `a`, `b`, `eps`, la fonction f et sa fonction dérivée, et renvoyant en sortie une valeur approchée à `eps` près calculée par la méthode de Newton de l'équation (E) , ainsi que le nombre n d'itérations pour l'obtenir.

b) Comparer le résultat pour $\texttt{eps} = 10^{-7}$ à celui obtenu par la méthode de dichotomie.

[Q6.] Soit $g : x \mapsto x^3 - x - 1$ sur $I = [0,3]$.

a) Appliquer 3 fois la méthode de Newton à g en prenant pour point initial $x_0 = 1$, puis $x_0 = 0,6$, et enfin $x_0 = 0,57$ pour obtenir une valeur approchée à 10^{-6} près de la solution dans I à l'équation $g(x) = 0$ en précisant le nombre d'itérations réalisées dans chaque cas (penser à programmer une boucle pour les 3 cas).

b) Comment expliquer que la méthode soit si sensible au choix de la condition initiale x_0 ?

Thème 3 : Notion de fonctions négligeables, équivalentes

I Préliminaires mathématiques

Soit x un nombre entier, et $\mathrm{ND}(x)$ le nombre de chiffres dans l'écriture (en base 10) de x. Par exemple : $\mathrm{ND}(1) = 1$, $\mathrm{ND}(87) = 2$, *etc*.

[Q1.] Rappeler la définition de la partie entière d'un réel x, notée $\lfloor x \rfloor$.

[Q2.] **a)** Soit $n \in \mathbf{N}^\star$ fixé. Donner deux réels α_n et β_n tels que l'on ait :

$$\mathrm{ND}(x) = n \iff \alpha_n \leq x < \beta_n$$

b) On rappelle que le logarithme décimal d'un réel u, noté $\log u$, est, par définition : $\log u = \dfrac{\ln u}{\ln 10}$. Déduire de ce qui précède une expression à l'aide des fonctions usuelles de la fonction ND. Dorénavant, à l'aide de cette expression, on pourra considérer la fonction ND définie sur $\mathbf{R}_+^\star$.

[Q3.] Programmer la fonction `ndigit(x)` qui prend en entrée un flottant x supposé strictement positif et qui renvoie en sortie l'entier $\mathrm{ND}(x)$.

[Q4.] Expliquer à l'aide de la fonction ND en quoi le logarithme est une fonction à croissance lente.

II Croissances comparées, équivalents

II – A Échelle en $+\infty$

On considère $x \in \mathbf{R}$, x destiné à tendre vers $+\infty$. On rappelle que :

a) Toute fonction f telle que $\lim\limits_{x \to +\infty} |f(x)| = +\infty$ s'appelle *infiniment grand* au voisinage de $+\infty$.

b) Toute fonction f telle que $\lim\limits_{x \to +\infty} |f(x)| = 0$ s'appelle *infiniment petit* au voisinage de $+\infty$.

[Q5.] Rappeler la définition de $f(x) = o(g(x)) \quad x \to +\infty$ pour deux fonctions f et g à valeurs réelles définies au voisinage de $+\infty$, ainsi que l'échelle de croissance en $+\infty$.

Dans ce qui suit, on travaillera avec des infiniment grands pour se fixer les idées, mais les résultats mis en évidence seraient également valables en analysant des infiniment petits.

[Q6.] Définir en python les fonctions `id`, `ln`, `sq` renvoyant (pour un flottant `x` en entrée donné) respectivement les flottants `x`, $\mathtt{x}^2$, $\ln \mathtt{x}$.

[Q7.] Recopier et expliquer ce que fait le script suivant :

```
 print("{0:5}|{1:22}|{2:22}|{3:22}|{4:22}".format("x=",
                                                  "ln x",
                                                   "sqrt x",
                                                   "x**2",
                                                   "exp x"))
for x in range(1,50):
    print("{0:5}|{1:22}|{2:22}|{3:22}|{4:22}".format(id(x),
                                                     int(ln(x)),
                                                     int(np.sqrt(x)),
                                                     int(sq(x)),
                                                     int(np.exp(x))))
```

II – B Équivalents

[Q8.] Soit f et g deux infiniments grands. Rappeler la définition de $f(x) \underset{x\to+\infty}{\sim} g(x)$.

La notion d'équivalent permet de donner un ordre de grandeur. Intuitivement, on peut penser que f et g sont équivalents signifie que les infiniment grands f et g ont, pour x assez grand [1], le même premier chiffre significatif (et le même ordre de grandeur bien entendu, c'est-à-dire le nombre de chiffres).

[Q9.] Écrire une fonction `first_digit(x)` qui prend en entrée un flottant `x` et qui renvoie en sortie l'entier égal à la valeur du premier chiffre de l'écriture (en base 10) de `x`. Par exemple : `first_digit(87431.1943)` devrait renvoyer 8. Il sera commode de convertir d'abord `x` en chaîne de caractères.

[Q10.] Écrire une fonction `synthese(x)` qui prend en entrée un flottant `x` et renvoie en sortie le tuple $\mathtt{n},\mathtt{p} = (\mathtt{first_digit(x)}, \mathtt{ND(x)})$

Intuitivement, si `synthese(x)` renvoie $(\mathtt{n},\mathtt{p})$, cela signifie que x est de l'ordre de $\mathtt{n} \times 10^{\mathtt{p}-1}$

[Q11.] Recopier et dire ce que fait le script suivant :

```
def equivalents(f,g,N=13):
    for p in range(1,N):
        x = 10**p
        print('x =',x)
        print(synthese(f(x)))
        print(synthese(g(x)))
        print("diff")
        print(f(x)-g(x))
        print('\n')
```

Puis dans la console :

```
>>> equivalents(f1,g1)
```

où vous aurez au préalable défini en python $f_1(x) = x^3 + 2x^2$ et $g_1(x) = x^3$.

[Q12.] Est-ce que $f(x) \underset{+\infty}{\sim} g(x) \Rightarrow f(x) - g(x) \to 0$?

[Q13.] Utiliser la fonction `equivalents` pour les couples de d'infiniment grands $(f(x), g(x))$ qui suivent afin de dire si ces dernières sont équivalentes ou non en $+\infty$:

1. C'est ce que signifie asymptotiquement

a) $(\ln x, \sqrt{x})$.

b) $(x^3 + 2x, x^3 + x)$.

c) $(x, 2x)$. Peut-on sommer les équivalents ?

d) $\left(\exp\left(x + \sqrt{x}\right), \exp x\right)$ (appliquer la fonction `equivalents` avec `N=3`) .Peut-on composer les équivalents ?

e) $(\sqrt{x^4 + x^3}, x^2)$ (appliquer la fonction `equivalents` avec `N=5`). Calculer mathématiquement l'équivalent.

f) $(\ln(1 + e^x), x)$.

Thème 4 : Intégration numérique

```python
import numpy as np
import matplotlib.pyplot as plt
```

I Principe de l'intégration numérique

Le point de départ des méthodes d'intégration est la formule suivante : si f est une fonction continue sur un segment $[a,b]$, $n \in \mathbf{N}^*$, et pour $k = 0, \ldots, n-1$, on pose : $x_k = a + kh$ où $h = \dfrac{b-a}{n}$, alors pour toute primitive F de f sur $[a,b]$:

$$F(b) - F(a) = \sum_{k=0}^{n-1} F(x_{k+1}) - F(x_k) = \sum_{k=0}^{n-1} \underbrace{\int_{x_k}^{x_k+h} f(t)\mathrm{d}t}_{I_k}.$$

II Méthode des rectangles

La méthode des rectangles repose sur l'interprétation géométrique de l'intégrale. Il y a plusieurs méthodes des rectangles, notamment :

a) La méthode des rectangles **à gauche**, qui consiste à approximer I_k en :

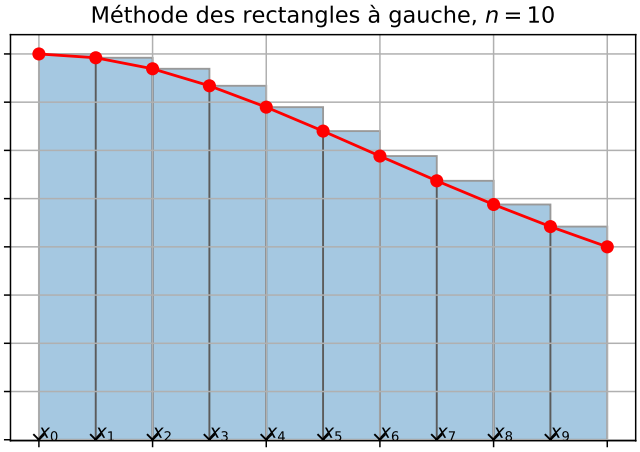
$$I_k \simeq R_G \quad R_G = f(x_k)h$$

b) la méthode des rectangles **à droite**, qui consiste à approximer I_k en :

$$I_k \simeq R_D \quad R_D = f(x_{k+1})h$$

c) Enfin, citons la méthode des **trapèzes**, qui consiste à faire une approximation moyenne à partir des deux méthodes précédentes en partant de l'approximation :

$$I_k \simeq \frac{R_G + R_D}{2}$$



[Q1.] Dessiner sur un schéma semblable au précédent ce que donneraient les méthodes précédentes sur seulement un intervalle $[x_K, x_{k+1}]$, et expliquer les dénominations retenues pour chacune des méthodes.

[Q2.] Programmer une fonction `Riemann(f,a,b,n)` qui prend en entrée une fonction f, les extrémités a, b d'un segment, un entier n et renvoie une valeur approchée de $\int_a^b f(t)\mathrm{d}t$ par la méthode des rectangles à gauche.

[Q3.] **a)** Essayer la fonction Riemann avec $f : x \mapsto x$ sur $[0,1]$ avec n de plus en plus grand pour vérifier que l'on obtient un résultat voisin de $1/2$.

b) Que se passe-t-il si on essaie `Riemann(f,1,0,n)` ?

[Q4.] Tester la fonction `Riemann` pour la fonction $g : x \mapsto \dfrac{4}{1+x^2}$ sur l'intervalle $[0,1]$ pour $n = 10^k$ $(1 \leq k \leq 6)$. Que dire de la vitesse de convergence ?

[Q5.] Comparer les valeurs obtenues par les fonctions `RG` et `RD` pour différentes valeurs de $h = 10^{-p}$, $p = 1, 2, \ldots 6$ sur la fonction g.

[Q6.] Il arrive que la méthode des rectangles soit définie non pas par le nombre de points n du segment d'intégration $[a, b]$, mais plutôt par son pas $h > 0$ (qui n'est donc plus exprimé en termes de a, b).

a) Programmer une fonction `RG(f,a,b,h)` qui donne une valeur appochée de $\int_a^b f(t)\mathrm{d}t$ par la méthode des rectangles à gauche avec un pas $h > 0$ spécifié en entrée. On pourra penser à utiliser la commande `np.arange`.

b) Que se passe-t-il si on essaie `RG(f,1,0,n)` ? Utiliser la fonction `RG` pour créer une fonction `RG2` tenant compte du cas `a>b`.

[Q7.] Programmer de même une fonction `RD(f,a,b,h)` qui donne une valeur appochée de $\int_a^b f(t)\mathrm{d}t$ par la méthode des rectangles à droite.

[Q8.] On considère la fonction définie sur **R** par :

$$F(x) = \int_x^{2x} \frac{1}{\sqrt{t^4+t^2+1}}\mathrm{d}t.$$

Tracer la courbe de cette fonction sur $[-10, 10]$. La courbe obtenue est celle d'une fonction impaire !

[Q9.] Soit F la fonction définie sur $\mathbf{R}_+^\star$ par :

$$F(x) = \int_0^1 \frac{\sin(tx)}{x+t} \mathrm{d}t.$$

Tracer la courbe de F sur $]0, 30[$.

Indication : on peut programmer soit une fonction `g(t,x)` *renvoyant en sortie le* flottant $\dfrac{\sin(tx)}{x+t}$, *soit la* fonction $J : x \mapsto \left(t \mapsto \dfrac{\sin(tx)}{x+t} \right)$ (ainsi $J(x)$ est elle-même une fonction).

Thème 5 : Méthode d'Euler et équations différentielles

```python
import numpy as np
import matplotlib.pyplot as plt
```

I Interprétation géométrique d'une équation différentielle

Considérons à titre d'exemple l'équation différentielle suivante avec condition initiale :

$$(E_0) \quad \begin{cases} y' &= y \\ y(0) &= 1 \end{cases}$$

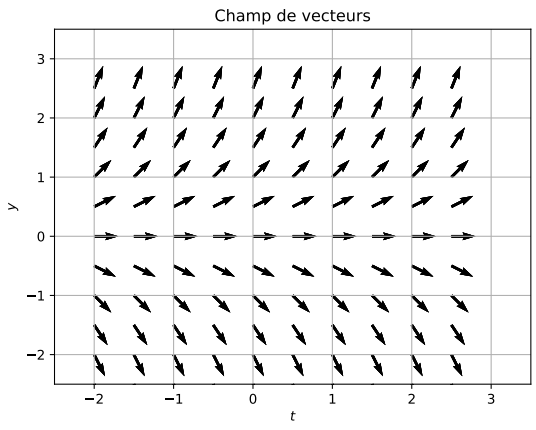
Afin de comprendre géométriquement ce que signifie cette équation différentielle, on l'étudie sans chercher à calculer exactement sa solution y_s.

[Q1.] **a)** Quelle est la valeur de $y_s'(0)$ d'après (E_0) ?

b) Tracer alors sur un dessin la direction de la tangente à la courbe de la solution y_s au point $(0, y_S(0))$.

c) De façon générale, si f est une fonction vérifiant l'équation $y' = y$, et si à une date t donnée, la valeur de $f(t)$ est connue, quelle est la pente de la tangente à à la la courbe C_f au point $(t, f(t))$?

d) Que représente le dessin suivant (appelé *champ des tangentes* ou *champ de vecteurs* associé à l'équation) ?



[Q2.] Compléter la phrase suivante qui est une reformulation de résolution d'une équation différentielle avec condition initiale :

Étant donné un champ de vecteurs associé à une équation différentielle, trouver une solution, c'est déterminer une ……dont les ……sont prescrites, partant d'un ……initial sur la ……

II Équation différentielle scalaire d'ordre 1

II – A Principe succintement rappelé

On fixe un intervalle $I = [a, b]$ et on se donne une équation différentielle (E) d'ordre 1 sur I avec condition initiale $y(a) = y_0$. La méthode d'Euler consiste, pour n un entier naturel non nul fixé, à calculer par récurrence et de façon approchée les valeurs y_k de la solution y de (E) aux points de I de la forme $t_k = a + kh$ pour $k = 0 \dots n$ où on a posé $h = \dfrac{b-a}{n}$. Ainsi :

$$(0) \quad \forall k \in \{0 \dots n\} \quad \text{on devrait avoir } y_k \simeq y(t_k) \quad \text{pour } h > 0 \quad \text{assez petit.}$$

En pratique, les valeurs approchées y_n se calculent par récurrence.

Précisément, on considère l'équation différentielle (E) sur un intervalle I avec condition initiale en $t_0 = a$:

$$(S) \quad \forall t \in I \quad \begin{cases} y' &= F(t, y(t)) \\ y(t_0) &= y_0 \end{cases}$$

[Q3.] En partant du taux d'accroissement à gauche d'une fonction, justifier l'approximation suivante :

$$y(t_{k+1}) \simeq y(t_k) \ + hF(t_k, y_k)$$

Dorénavant, on remplacera la résolution du système (S) par le calcul des premiers termes de la suite récurrente (y_n) définie par :

$$(S') \quad \forall n \in \mathbf{N} \quad \begin{cases} y_0 & \text{donné} \\ y_{k+1} &= y_k + hF(t_k, y_k) \end{cases}$$

Le système (S') s'appelle discrétisation de (S) par la méthode d'Euler à gauche.

[Q4.] **a)** Remplir le tableau ci-dessous pour l'équation (E_0), pour $I = [0, 3]$, et $n = 6$:

$k =$	$t_k =$	$y_k =$	$F(t_k, y_k) =$	$y_s'(t_k) =$
1				
2				
3				
4				
5				
$6 (= n)$				

b) En quoi consiste alors géométriquement la méthode d'Euler à gauche ?

c) Pouvez vous expliquer intuitivement en quoi le principe (0) semble raisonnable à partir de l'interprétation géométrique d'une équation différentielle ?

On notera que la solution y à (S) devient donc dans le schéma à une suite finie de valeurs approchées de cette dernière sur l'intervalle $[t_0, t_n]$.

[Q5.] Programmer une fonction `EulerG(F,y0,a,b,n)` qui prend en entrée une fonction F, une condition initiale y_0, les extrémités a, b d'un intervalle I, un entier $n > 0$ et qui renvoie en sortie les listes des points `[t0,..,tn]` (où $t_k = a + kh$) et `[y0,...,yn]` obtenues par discrétisation du système (S) en le système (S') (on rappelle que $h = \dfrac{b-a}{n}$).

II – B Étude théorique d'un exemple historique

Soit $T > 0$. On revient à l'équation différentielle sur $I = [0, T]$ suivante :

$$(E_0) \quad \begin{cases} y' &= y \\ y(0) &= 1 \end{cases}$$

Remarque. Cette équation s'appelle aussi équation de Malthus.

[Q6.] **a)** Donner la solution exacte y_e de (E_0).

b) Écrire le système (S') correspondant à l'équation (E_0).

c) Résoudre explicitement le système (S').

Pour vérifier le principe (0), à savoir que lorsque le pas de discrétisation h tend vers 0, la solution approchée devrait «converger» vers y_e, on procède de la manière suivante : on fixe un entier $N > 0$, et on pose $h = T/N$. D'après (0), $y_N \simeq y_e(t_N) = y_e(t)$, donc si $N \to \infty$, (c'est-à-dire $h \to 0$), on devrait avoir $y_N \to y_e(T)$.

[Q7.] Montrer que $y_N \to e^T$ quand $N \to \infty$. On cherchera pour cela, un équivalent de $v_N = \ln(1 + T/N)$ quand $N \to \infty$.

[Q8.] **a)** Utiliser la fonction `EulerG` pour tracer sur un même graphique une solution approchée de (E_0) sur l'intervalle $I = [0, 3]$ avec un pas $n = 30$, ainsi que la courbe de la fonction y_L. Noter que graphiquement, la solution approchée est par définition une ligne brisée.

b) Répéter ce tracé pour des valeurs de n à chaque fois 10 fois plus grandes et constater.

II – C Étude d'un second exemple

On considère l'équation différentielle logistique sur $I = \mathbf{R}_+$ suivante :

$$(L) \quad \begin{cases} y' &= y\left(1 - \dfrac{y}{10}\right) \\ y(0) &= 0.5 \end{cases}$$

On admet que cette équation différentielle admet une unique solution y_L sur $\mathbf{R}_+$.

[Q9.] notations du système (S), expliciter la fonction F pour ce système (L), ce dernier étant bien un cas particulier du système général (S).

[Q10.] Même question que `[Q8.]` pour (L) au lieu de (E_0), sachant que la solution exacte de (L) sur I est :

$$y_L : t \mapsto \frac{10}{1+19e^{-t}}$$

II – D Un exemple d'équation non autonome

[Q11.] On considère l'équation différentielle suivante sur $\mathbf{R}_+$:

$$(G) \quad \begin{cases} y' + 2ty &= 0 \\ y(0) &= 1 \end{cases}$$

Écrire le schéma d'Euler pour cette équation différentielle est calculer une solution approchée sur l'intervalle $[0, T]$ ($T > 0$ donné) avec la fonction `EulerG`.

III Système d'équations différentielles scalaires

On peut réutiliser les idées de la méthode d'Euler pour les appliquer au cas de systèmes différentiels. Dans ce dernier cas, il y a plusieurs fonctions inconnues, donc plusieurs suites à programmer.

III – A Contexte

On considère les équations cinétiques suivantes d'ordre 1 mettant en jeu des substances A,B,C : Les équations cinétiques sont les suivantes :

$$\begin{cases} \dfrac{\mathrm{d[A]}}{\mathrm{dt}} &= -k_1[\mathrm{A}] \\ \dfrac{\mathrm{d[B]}}{\mathrm{dt}} &= k_1[\mathrm{A}] - k_2[\mathrm{B}] \\ \dfrac{\mathrm{d[C]}}{\mathrm{dt}} &= k_2[\mathrm{B}] \end{cases}$$

avec les conditions initiales suivantes à $t = 0$:

$$[\mathrm{A}]_0 = a > 0, \quad [\mathrm{B}]_0 = [\mathrm{C}]_0 = 0.$$

[Q12.] Écrire le schéma d'Euler à gauche pour ce système différentiel. On notera que cette fois, une solution du système différentiel est un triplet de trois fonctions $([\mathrm{A}], [\mathrm{B}], [\mathrm{C}])$.

[Q13.] Résoudre ces équations par la méthode d'Euler à gauche et représenter les concentrations [A], [B], [C] en fonction du temps sur l'intervalle de temps $[0, 6s]$ dans les cas suivants (les constantes de réaction sont en s^{-1}) :

a) $(k_1, k_2) = (1, 1)$.

b) $(k_1, k_2) = (10, 1)$.

c) $(k_1, k_2) = (1, 10)$.

On prendra $a = 1\text{mol.L}^{-1}$

On écrira par exemple une fonction `Euler2(a,k1,k2,n)` et on renverra en sortie les listes `Lt`,`La`, `Lb`,`Lc` déduites par le schéma d'Euler.

IV Oscillateur harmonique non amorti

IV – A Présentation du modèle

On s'intéresse à un oscillateur harmonique (non amorti) régi par l'équation différentielle :

$$(1) \qquad \frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -\omega^2 x.$$

Pour étudier commodément cette équation différentielle, on introduit une seconde coordonnée : $y = \frac{1}{\omega}\frac{dx}{dt}$ (y est proportionnelle à la vitesse). On peut traduire l'équation (1) sous la forme

$$(E) \qquad \begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \omega y \\ \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = -\omega x \end{cases}$$

Le plan repéré par les coordonnées (x, y) s'appelle l'*espace des phases* associé à l'équation (1). Tout point de ce plan s'appelle un état du système.

Une solution du système (E) est un couple de fonctions (x, y) définies et de classe $\mathscr{C}^1$ sur $[0, +\infty[$ et telles que

$$\forall t \in [0, +\infty[, \begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = \omega y(t) \\ \dfrac{\mathrm{d}y}{\mathrm{d}t} = -\omega x(t) \end{cases}$$

Pour un état (u, v) de l'espace des phases, l'énergie est définie par $E(u, v) = u^2 + v^2$. Cela signifie que si à un instant donné, le système est dans l'état (u, v) (*i.e.* $x(t) = u$, et $y(t) = v$), son énergie à cet instant est égale à $E(u, v)$.

[Q14.] Soit $t \mapsto (x(t), y(t))$ une solution de (E). Par quelle propriété mathématique sur la fonction $t \mapsto E(x(t), y(t))$ se traduit la conservativité du système ?

IV – B Schéma d'Euler explicite

[Q15.] Vérifier que la définition du taux d'accroissement à gauche conduit aux approximations suivantes pour (E) :

$$x\left(t_{n+1x}\right) \approx x\left(t_n\right) + h\omega y\left(t_n\right)$$
$$y\left(t_{n+1}\right) \approx y\left(t_n\right) - h\omega x\left(t_n\right)$$

et qu'il est cohérent avec cette approximation de définir une suite de points de l'espace des

phases $((x_n, y_n))_{n\geq 0}$ par :

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = M \begin{pmatrix} x_n \\ y_n \end{pmatrix} \quad \text{où} \quad M = \begin{pmatrix} 1 & h\omega \\ -h\omega & 1 \end{pmatrix}.$$

Cette relation de récurrence s'appelle discrétation de (E) par schéma d'Euler explicite de pas h.

On notera dans la suite par commodité pour tout entier k :

$$X_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} \quad \text{si bien que} \quad X_{n+1} = MX_n.$$

En outre on prendra une condition initiale de $(x(0), y(0)) = (a, 0)$, avec $a > 0$.

[Q16.] À quelle situation correspond ce choix de X_0 dans le modèle ?

Pour toutes les simulations numériques qui suivent, on pourra fixer $\omega = 2\pi$

[Q17.] On fixe un instant final $T > 0$ et on examine les solutions sur $I = [0, T]]$.

a) Créer une fonction appelée `euler_exp` prenant en entrée `x0,y0`, un entier positif `N`, et qui renvoie les listes `[x_0,...,x_N],[y_0,...,y_N]` issues de l'application de la méthode d'Euler explicite à partir de (x_0, y_0) avec N itérations sur l'intervalle I.

b) Tester la fonction `euler_exp` pour des conditions initiales $(a, 0)$ de votre choix. On pourra pour cela tracer la famille de points `(x[k],y[k])`, avec par exemple `N = 100` et faire varier N.

[Q18.] **a)** Créer une fonction appelée `euler_exp2` prenant en entrée les mêmes arguments que la fonction `euler_exp` qui renvoie en sortie la liste des valeurs de l'énergie $E_k = E((x[k], y[k]))$ aux points `(x[k],y[k])` obtenus par `euler_exp` pour `k` allant de `0` à `N`.

b) Représenter graphiquement les valeurs de E_k en fonction de t_k obtenues par `euler_exp2` avec les données initiales que vous aviez choisies.

c) Montrer par un calcul que $E(x_{x+1}, y_{k+1}) = (1 + h^2\omega^2)^k E(x_0, y_0)$. Que penser du modèle vis-à-vis de la propriété de conservation de l'énergie du système ?

IV – C Schéma d'Euler implicite

[Q19.] Vérifier que la définition du taux d'accroissement à droite conduit à la discrétisation par schéma d'Euler (dite implicite) suivante :

$$Q \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \end{pmatrix} \quad \text{où cette fois} \quad Q = \begin{pmatrix} 1 & -h\omega \\ h\omega & 1 \end{pmatrix}$$

Remarque. Dans ce dernier schéma, X_{n+1} ne se calcule pas explicitement à partir de X_n : la détermination de X_{n+1} passe par la résolution d'une équation. Ce qui explique le qualificatif *implicite* dans la méthode.

[Q20.] Créer une fonction appelée `euler_imp` prenant en entrée `x0, y0`, le pas `h` et un entier positif `N`, et qui renvoie les listes `[x_0,...,x_N]`, `[y_0,...,y_N]` issues de l'application de la méthode

d'Euler explicite à partir de (x_0, y_0) avec N itérations. Tester sa fonction avec un point initial de l'espace des phases de la forme $(a, 0)$. On pourra au préalable de la programmation calculer la matrice Q^{-1} pour produire un script semblable à celui de `[Q11.]`.

[Q21.] Créer une fonction appelée `euler_imp2` prenant en entrée `x0, y0`, le pas `h` et un entier positif `N`, et qui renvoie la des valeurs de l'énergie E sur les points de l'espace des phases $(\mathtt{x}_0, \mathtt{y}_0), \ldots, (\mathtt{x}_N, \mathtt{y}_N)$ obtenus par appel de la fonction `euler_imp`. Tester sa fonction avec un état initial $(\mathtt{x0}, \mathtt{y0})$ de l'espace des phases de la forme $(a, 0)$.

[Q22.] Montrer par un calcul que $E(x_{x+1}, y_{k+1}) = \dfrac{1}{(1+h^2\omega^2)^k} E(x_0, y_0)$. Que penser du modèle vis-à-vis de la propriété de conservation de l'énergie du système ?

IV – D Schéma d'Euler trapézoïdal

[Q23.] Vérifier que la définition du taux d'accroissement centré conduit à la discrétisation suivante de l'équation (E)

$$\forall n \in \mathbf{N} \quad \begin{pmatrix} 1 & -\dfrac{h\omega}{2} \\ \dfrac{h\omega}{2} & 1 \end{pmatrix} X_{n+1} = \begin{pmatrix} 1 & \dfrac{h\omega}{2} \\ -\dfrac{h\omega}{2} & 1 \end{pmatrix} X_n \quad \text{et} \quad X_0 = \begin{pmatrix} a \\ 0 \end{pmatrix} \quad \text{où} \quad a > 0.$$

[Q24.] Refaire la même étude que pour le schéma d'Euler explicite et conclure sur cette discrétisation.

Thème 6 : Suites réelles

```python
import numpy as np
import matplotlib.pyplot as plt
```

I Étude d'une suite implicite

Pour tout entier $n \geq 2$, on considère la fonction f_n défnie sur $I = \mathbf{R}_+$ par :

$$f_n(x) = \frac{x^n}{\sqrt{1+x^2}},$$

et on considère pour $n \geq 3$ l'équation :

$$(E_n) \quad f_n(x) = 1.$$

[Q1.] Justifier que l'équation (E_n) possède une unique solution dans I, notée x_n.

[Q2.] Tracer la courbe de la fonction f_n pour les premières valeurs de n afin de trouver un segment $[a, b]$ d'extrémités entières indépendantes de n contenant x_n pour $n \geq 2$.

[Q3.] **a)** Écrire une fonction `xn(n,eps)` qui prend en entrée un entier n, un flottant `eps` et qui renvoie en sortie une valeur approchée de x_n à `eps` près obtenue par dichotomie sur l'intervalle $[a, b]$ déterminé à la question précédente.

b) Donner la liste des 15 premiers termes de la suite (x_n) à 10^{-5} près. Conjecturer la montonie et la nature de la suite (x_n).

II Étude d'une suite récurrente

II – A Fonctions de base

La suite logistique (vue en cours dans le chapitre de dynamique des populations) est la suite définie par :

$$\begin{cases} u_0 \in [0,1] \\ \forall n \in \mathbf{N} \qquad u_{n+1} = \mu u_n (1-u_n) \end{cases} \tag{6.1}$$

où μ est un paramètre dans $[0, 4]$.

Le modèle logistique est une correction du modèle malthusien (par le facteur $1-u_n$). Pour autant, le comportement de la suite ainsi définie est très variable suivant les valeurs de μ et u_0, et peut se montrer extrêmement compliqué.

[Q4.] Programmer la fonction f_μ associée à cette suite récurrente.

[Q5.] Tracer sur l'intervalle $[0,1]$ la courbe de la fonction $f_{0.75}$ par exemple. On pourra commencer par créer la liste `X` contenant 100 points régulièrement répartis de 0 à 1, puis la liste `Y` des images par $f_{0.75}$ en compréhension.

[Q6.] Écrire une fonction `listeValeurs(mu,u0,n)` prenant en entrée deux flottants `mu`, `u0`, ainsi qu'un entier `n` et retournant en sortie la liste `L` des valeurs $u_0, \dots u_n$ des termes de la suite définie par (6.1). Testez votre fonction pour $\mu = 0,5$, $u_0 = 0,3$, $n = 5$.

II – B Représentation graphique

Une façon classique de représenter graphiquement les termes de la suite est de considérer cette dernière comme une fonction définie sur $\mathbf{N} \subset \mathbf{R}$.

[Q7.] Tracer graphiquement la famille de points $\Big((n,u_n)\Big)_{0\leq n\leq 20}$ afin d'observer le comportement de la suite. On pourra essayer par exemple les valeurs de μ et u_0 données dans le tableau ci-dessous et conjecturer la monotonie et nature de la suite :

μ	u_0
0,81	0,1
	0,83
1,93	0,02
	0,75
2,75	0,43
2,81	0,46
3,45	0,23
3,67	0,23

[Q8.] **a)** Écrire une fonction qui prend en entrée une liste `L` et qui renvoie en sortie le tuple `Lp,Li` des sous-listes des termes de `L` d'indices respectivement pairs et impairs. Penser à un slicing sur les listes

b) Tracer pour les deux derniers sets de valeurs (μ, u_0) les sous-suites des termes de rang pair et impair des suites (u_n) correspondantes.

II – C Obtention du diagramme de bifurcation

[Q9.] L'instruction `round(a,n)` arrondit un flottant `a` à `n` décimales. Écrire une fonction `asymptotique(mu,u0)` qui prend en entrée les flottants `mu` et `u0` et renvoie en sortie la liste des termes $u_{501}, \dots, u_{600}$ arrondis à trois décimales. On pensera à utiliser la fonction `listeValeurs` et réaliser une extraction de sous-liste par *slicing*.

[Q10.] Appliquer cette dernière fonction pour $u_0 = 0,75$ et les valeurs $\mu \in \{1.1, 2.1, 3.1, 3.5\}$, mais n'afficher que les 8 dernières valeurs de chaque liste obtenue. Que remarque-t-on ?

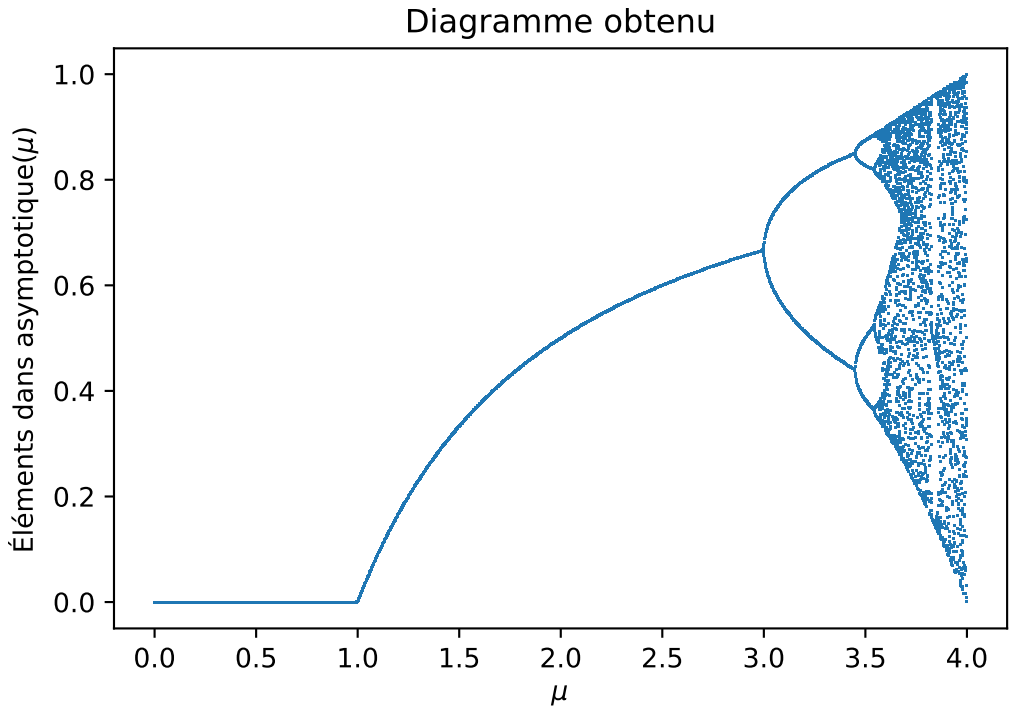
[Q11.] Écrire une fonction `def image(liste):` qui prend en entrée une liste de flottants et retourne en sortie la liste `V` des valeurs distinctes de la liste. Par exemple, si `liste` = `[1,2,3,1,2,3,1,1,1]`, la fonction devrait renvoyer la valeur `[1,2,3]`.

[Q12.] On aimerait, pour u_0 fixé, savoir combien de valeurs distinctes se trouvent dans la liste `asymptotique(mu,u0)` en fonction de μ.

On fixe ici $u_0 = 0,75$. Répondre à cette question en traçant graphiquement la fonction qui à μ associe ce nombre. On pourra faire varier μ de 0 à 4 par pas de $1/100$ par exemple. On tracera le nuage de points en spécifiant dans la commande `plot` l'option : `","`

[Q13.] Dans cette question, on prend encore $u_0 = 0.75$. Tracer un graphique sur lequel figurent en abscisses les valeurs de $\mu \in [0, 4]$ et en ordonnées les valeurs se trouvant dans les différentes listes `image(asymptotique(mu,0.75))` (ainsi par exemple, si la liste `image(asymptotique(mu,0.75))` contient trois éléments, sur le diagramme, on dessinera 3 points à l'abscisse `mu`). On fera varier μ par pas de `0.002`.

Vous devriez obtenir un diagramme de ce genre :



Remarque. Le diagramme obtenu s'appelle *diagramme de bifurcations* de la suite logistique.

Thème 7 : Polynômes

```
1 import numpy as np
2 N = 10 # Constante. On travaille dans IR_N[X]
```

Dans tout le texte, on fixe un entier $N > 0$ et on travaille dans l'ensemble :

$$\mathbf{R}_N[X] = \{P \in \mathbf{R}[X] \quad \deg(P) \leq N\}$$

(*cf.* ligne 2 du script précédent).

I Choix de la structure de données

a) Si $P = a_N X^N + \cdots + a_1 X + a_0$ est un polynôme à coefficients réels de degré au plus N, on décide de le représenter en python par la liste `L` de longueur `N+1` de ses coefficients rangés par *degré croissant* : `L = [a0,...,aN]`.

b) Par exemple, dans $\mathbf{R}_2[X]$, :

— si $P = X^2 - 3X + 2$, sa représentation sous python est `[2,-3,1]`.

— si $P = X - 3$, sa représentation sous python est `[-3,1,0]`.

[Q1.] Écrire une fonction `RN(L,d=N)` (la variable `d` prend ainsi par défaut la valeur `N` déclarée au début du script) prenant en entrée une liste de flottants `L` et qui renvoie en sortie la liste `L` à laquelle on concatène éventuellement à droite une liste de `0` de sorte que la liste de sortie soit de longueur exactement `d+1`. En particulier, si la longueur de `L` excède `N+1`, la fonction renvoie la liste `L`. Par exemple, `RN([1,2],3)` renvoie `[1,2,0,0]`, mais `RN([1,2])` renvoie `[1,2,0]` si on avait fixé `N=2`. Enfin `RN([1,2,2,2,2],3)` renvoie `[1,2,2,2,2]`.

[Q2.] Écrire une fonction `deg(P)` qui prend en entrée une liste de longueur `N+1` représentant un polynôme $P \in \mathbf{R}_N[X]$ et qui :

— renvoie en sortie le degré de P si $P \neq 0$.

— renvoie `-1` si P est nul (on convient que le polynôme nul est de degré -1 sous python).

[Q3.] Écrire une fonction `eval(P,a)` prenant en entrée une liste de flottants représentant un polynôme P, un flottant `a`, et renvoyant en sortie la valeur du réel $P(a)$.

II Opérations sur les polynômes

II – A Multiplication par un scalaire

[Q4.] Écrire une fonction `dot(alpha,P)` qui prend en entrée un flottant `alpha`, une liste représentant polynôme P et renvoyant en sortie la liste représentant le polynôme αP.

II – B Somme

[Q5.] Écrire une fonction `somme1(P,Q)` qui prend en entrée deux listes représentant des polynômes P et Q de $\mathbf{R}_N[X]$, et renvoyant en sortie la liste représentant $P+Q$.

[Q6.] Écrire une fonction `somme2(P,Q)` qui prend en entrée deux listes représentant des polynômes P et Q de degré quelconque, et renvoyant en sortie la liste représentant $P+Q$ vue comme un élément d'un espace $\mathbf{R}_d[X]$ adéquat.

II – C Produit

[Q7.] Écrire une fonction `produit(P,Q)` qui prend en entrée deux listes représentant des polynômes P et Q, et renvoyant en sortie la liste représentant PQ à l'aide de la formule donnant le produit de deux polynômes (dans un espace $\mathbf{R}_d[X]$ adéquat).

[Q8.] (*Autre manière de programmer le produit*).

a) Soit $k \in \mathbf{N}$. Quel est l'effet de la multiplication par le monôme X^k sur [la liste des coefficients d'] un polynôme P ?

b) Programmer une fonction `shift(k,L)` qui prend en entrée un entier `k`, une liste de flottants `L` et renvoyant en sortie la liste obtenue par concaténation des listes $\underbrace{[0,\ldots,0]}_{k \text{ items}}$ et `L`.

c) Utiliser les fonctions `dot`, `shift` et `somme1` ou `somme2` pour programmer le produit de deux polynômes de $\mathbf{R}_N[X]$.

[Q9.] **a)** Soit $N \in \mathbf{N}$. Quels sont les coefficients du polynôme $(1+X)^N$?

b) Utiliser la fonction `produit` pour écrire une fonction `pascal(N)` donnant la liste des lignes 0 à $N-1$ du triangle de Pascal. Par exemple :

```
>>> pascal(5)
 [[1, 0, 0, 0, 0, 0],
  [1, 1, 0, 0, 0, 0],
  [1, 2, 1, 0, 0, 0],
  [1, 3, 3, 1, 0, 0],
  [1, 4, 6, 4, 1, 0]]
```

III Autour de la méthode de Hörner

III – A Présentation et programmation

Soit $P = a_n X^n + \cdots + a_1 X + a_0$ et $\alpha \in \mathbf{R}$

La méthode de Hörner est une méthode permettant d'économiser le nombre d'opérations dans le calcul de $P(\alpha)$ en remarquant que dans le calcul de $Q(\alpha)$ pour n'importe quel polynôme Q, tous les termes, à part le terme constant de Q peuvent se factoriser par α.

Plus explicitement, on effectue le calcul de $P(\alpha)$ en écrivant :

$$P(\alpha) = \Big(\Big(\big((a_n\alpha + a_{n-1})\alpha + a_{n-2}\big)\dots\Big)\alpha + a_1\Big)\alpha + a_0.$$

Précisément, l'expression ci-dessus se résume formellement de la manière suivante : si on définit pour tout entier k tel que $0 \leq k \leq n$ les fonctions affines f_k par :

$$x \mapsto f_k(x) = \alpha x + a_{n-k},$$

il se trouve que l'on a :

$$P(\alpha) = u_n$$

où les nombres $u_0, \dots, u_n$ sont définis par récurrence par :

$$(H) \quad \begin{cases} & u_0 &=& f_0(0) \\ \forall k \in \{0,\dots,n-1\} & u_{k+1} &=& f_{k+1}(u_k) = \alpha u_k + a_{n-k-1} \end{cases}$$

Remarque. La suite ainsi définie est une suite récurrente. Néanmoins, à chaque étape, le calcul d'un terme se fait en fonction du précédent à travers une fonction f_k dépendant du rang du terme calculé (alors que classiquement, on applique la même fonction f à tout rang).

[Q10.] Exemple : $n = 3$, et $P = a_3X^3 + a_2X^2 + a_1X + a_0$. Compléter le tableau suivant :

$f_0(x) =$		$u_0 =$	
$f_1(x) =$		$u_1 =$	
$f_2(x) =$		$u_2 =$	
$f_3(x) =$		$u_3 =$	

et vérifier que l'on a bien $u_3 = P(\alpha)$.

[Q11.] Programmer une fonction `horner(P,a)` qui prend en entrée un polynôme et qui renvoie en sortie la valeur de $P(a)$ calculée par la méthode de Hörner.

Pour comparer les temps de calcul entre `eval(P,a)` et `horner(P,a)`, on pourra choisir les polynôme et flottant P, a de son choix, et taper dans la console :

```
>> %timeit eval(P,a)
>> %timeit horner(P,a)
```

III – B Factorisation d'un polynôme par $X - \alpha$

Soit $n \in \mathbf{N}^*$ donné et $P = a_nX^n + \dots + a_1X + a_0$ ($a_n \neq 0$) et mettons que α soit racine de P, si bien que pour des coefficients $b_{n-1}, \dots, b_0$, on a : $a_nX^n + \dots + a_1X + a_0 = (X - a)(b_{n-1}X^{n-1} + \dots + b_1X + b_0)$

[Q12.] **a)** En procédant par identification, trouver une relation entre les nombres b_j et les nombres u_k definis par (H).

b) En déduire une fonction `factorise(P,a)` qui prend en entrée un polynôme P, un flottant a racine de P, et qui renvoie en sortie la liste des coefficients du polynôme Q tel que $P = (X-a)Q$.

[Q13.] Donner la factorisation par $X-1$ du polynôme représenté par :

```
[-69, 6, 7, 9, 1, 6, 2, 2, 7, 5, 6, 2, 1, 2, 4, 9].
```

Thème 8 : Récursivité

I Récursivité

I – A Présentation du concept

La récursivité est un concept important en informatique et permet parfois de coder de façon très élégante (mais pas toujours efficace) des fonctions. *Grosso modo*, une fonction est dite *récursive* si dans sa définition cette dernière s'appelle elle-même. La programmation récursive est une approche à opposer l'approche classique appelée *programmation itérative*.

Exemple. *Programmation de la factorielle d'un entier n.*

Programmation récursive

Partant du constat que $n! = n \times (n-1)!$, et sachant que $0! = 1$, la programmation récursive de la factorielle conduirait au code suivant :

```
def facto(n):
    if n==0:
        return 1
    else:
        return n*facto(n-1)
```

Programmation itérative

Une programmation itérative possible :

```
def facto(n):
    s = 1
    for k in range(n,1,-1):
        s*=n
    return s
```

I – B Ce qu'il faut repérer pour programmer récursivement

Afin d'être certain qu'une programmation récursive aboutisse, on peut s'appuyer sur le principe de descente de Fermat :

Toute suite d'entiers naturels strictement décroissante n'a qu'un nombre fini de termes.

Les ingrédients d'une programmation récursive réussie sont alors :

a) Identification d'une grandeur entière G associé à la variable d'entrée, et qui décroît strictement à chaque appel de la fonction programmé récursivement.

b) Existence d'une relation simple entre les objets à calculer pour les deux valeurs consécutives G et $G-1$.

c) Initialisation de la valeur de l'objet à calculer pour la valeur minimale de G.

À vous d'identifier G lors d'une programmation récursive.
Remarque. Dans la plupart des cas :

a) Si la fonction prend en argument un entier `n`, $G =$ `n`

b) Si la fonction prend en argument une liste `L`, $G =$ `len(L)`

c) Si la fonction prend en entrée une chaîne de caractères `c`, $G =$ `len(c)`

Exemple. Pour la fonction factorielle programmée précédemment :

a) la grandeur G est simplement n.

b) la relation liant les objets à calculer entre les deux grandeurs consécutives de G est $n! = n \times (n-1)!$, et donc à chaque appel de la fonction récursive, n décroît strictement.

c) Initialisation pour le minimum $G = 0$ par $0! = 1$: l'exécution du programme devrait donc terminer au plus en n appels.

[Q1.] Programmer récursivement une fonction `puissance(x,n)` prenant en entrée un flottant x et un entier n et renvoyant en sortie la valeur de x^n.

II Exercices

II – A Récursivité sur des calculs de flottants

[Q2.] Soit a, b deux réels et $(u_n)_{n\in\mathbf{N}}$ la suite définie par :

$$(S_{a,b}) \quad \begin{cases} u_0 \text{ donné} \\ \forall n \geq 0 \quad u_{n+1} = au_n + b \end{cases}$$

a) Programmer itérativement une fonction `ag(a,b,u0,n)` prenant en entrée 3 flottants `a`,`b`,`u0` et un entier `n`, donnant en sortie la valeur du terme u_n d'indice n de la suite définie par $(S_{1,b})$.

b) Même question mais avec une programmation récursive.

c) On pourra tester ses fonctions pour calculer u_{12} avec la suite définie par : $u_0 = 1$ et le système $\left(S_{-\frac{1}{2},4}\right)$ (on devrait trouver $u_{12} \simeq$ `2.666259765625`)

[Q3.] Soit f une fonction définie sur un sous-ensemble de $\mathbf{R}$, et (u_n) une suite récurrente définie par :

$$(S) \quad \begin{cases} u_0 \text{ donné} \\ \forall n \geq 0 \quad u_{n+1} = f(u_n) \end{cases}$$

Mêmes questions que précédemment : programmation récursive et itérative de la suite définie par (S). Ensuite, vérification avec par exemple pour $u_0 = \dfrac{1}{2}$ et $f(x) = \dfrac{16}{5}x(1-x)$. (*réponse* : $u_{12} \simeq$ `0.5130444051432489`)

[Q4.] La suite de Fibonacci est la suite$(F_n)_{n\geq 0}$ définie par :

$$(F) \quad \begin{cases} F_0 = F_1 = 1 \\ \forall n \geq 0 \quad F_{n+2} = F_{n+1} + F_n \end{cases}$$

a) Programmer itérativement une fonction `Fibo(n)` qui prend en entrée un entier `n` et qui renvoie en sortie le terme F_n de cette suite (F_n). Afficher ses termes jusqu'à F_{37}.

b) Programmer `Fibo2(n)`, version récursive de `Fibo`. Afficher les termes jusqu'à F_{37}. Conclusion ?

II – B Programmation récursive sur les objets structurés

On rappelle que `L1+L2` *concatène les listes* `L1` *et* `L2` *dans cet ordre. Même effet si* `L1` *et* `L2` *sont des chaines de caractères*.

[Q5.] **a)** Définir une fonction récursive `remplace(L)` prenant en entrée une liste `L` de flottants et renvoyant en sortie la liste `L` dans laquelle tous les nombres négatifs sont remplacés par des `0`. On pourra partir du fait que `L= [L[0]]+ L[1:]`.

b) Soit f une fonction définie sur un sous-ensemble de $\mathbf{R}$, et (u_n) une suite récurrente définie par :

$$(S_{u_0}) \quad \begin{cases} u_0 \text{ donné} \\ \forall n \geq 0 \quad u_{n+1} = f(u_n) \end{cases}$$

En remarquant :

— d'une part que `[u0..un]= [u0]+[u1...un]`,

— et d'autre part que le terme de rang $n-1$ de la suite définie par $(S_{f(u_0)})$ est bien le terme u_n de (S_{u_0}),

programmer récursivement une fonction prenant en entrée un entier `n` et renvoyant en sortie la liste des termes $u_0, \ldots, u_n$ de la suite définie par S_{u_0}

c) Programmer récursivement une fonction `maxi(L)` qui calcule le maximum d'une liste de flottants `L`. On partira du constat que `L=[L[0]]+L[1:]` et utilisera la fonction python `max` pour calculer le plus grand de deux flottants.

d) Programmer récursivement une fonction `noccur(L,e)` prenant en entrée une liste de flottants `L`, un flottant `e` et qui renvoie en sortie le nombre d'occurrences de l'élément `e` dans la liste `L`.

e) Programmer récursivement une fonction `replique(L,n)` prenant en entrée une liste `L`, un entier `n`, et renvoyant en sortie la liste dans laquelle la séquence constituant les éléments de `L` est répétée n fois.

f) Programmer récursivement une fonction `flip(L)` prenant en entrée une liste `L` et renvoyant en sortie la liste obtenue en renversant l'ordre des termes de `L`.

g) Programmer récursivement la méthode de Hörner pour l'évaluation $P(\alpha)$ d'un polynôme P en remarquant que si $P = a_0 + a_1 X + \cdots + a_n X^n$, :

$$P(\alpha) = a_0 + \alpha(Q(\alpha))$$

où Q est est un polynôme à préciser.

h) Définir une fonction récursive `remplace(chaine)` prenant en entrée une chaine de caractères `chaine` et renvoyant en sortie la chaine `chaine` dans laquelle les `e` sont remplacés par des `o`.

Thème 9 : Module numpy

```
1 import numpy as np
2 import numpy.linalg as la
```

Le module numpy est un module d'ingénierie mathématique conçu pour implémenter de manière efficace le calcul matriciel. Fondamentalement, une matrice est vue dans Python comme une liste, précisément, la *liste de ses lignes*. De plus, la déclaration de cette liste par le mot-clé array fait hériter à l'objet de nouvelles opérations en relation avec le calcul matriciel, et dont les listes ne disposent pas.

I Opérations de base

I – A Définition de matrices

Construisons la matrice :

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{pmatrix}.$$

```
1 ligne1 = [1,2,3,4]
2 ligne2 = [5,6,7,8]
3 liste = [ligne1,ligne2]
4 A =np.array(liste) # une matrice bidimensionnelle est une liste de
      lignes
5 print(A)
6 print(A.shape)
```

[Q1.] Que fait la commande A.shape ? Quel type d'objet renvoie-t-elle ?

On parle de tableau numpy bidimensionnel. Ce sont ces tableaux qui représentent les matrices de $\mathscr{M}_{n,p}(\mathbf{R})$.

[Q2.] Que renvoie la dernière commande ci-dessous ? Expliquer le résultat des commandes suivantes :

```
>>> X = np.arange(0,10,1)
>>> X.shape
```

[Q3.] Même question avec :

```
>>> X.reshape(10,1)
```

La commande reshape est utile pour définir des matrices rapidement.

[Q4.] Utiliser la commande `reshape` pour définir la matrice suivante :

$$B = \begin{pmatrix} 1 & 0 & -1 & 1 \\ -1 & 1 & 0 & -1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & 0 \end{pmatrix}$$

[Q5.] Taper dans la console successivement les trois instructions suivantes : `A[1,3]`, `A[0,0]`, `A[2,1]`. Que retenir de ceci ?

I – B Construction de matrices, matrices utiles

[Q6.] Fabriquer les matrices suivantes :

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 \end{pmatrix} \in \mathscr{M}_{1,5}, \quad C = \begin{pmatrix} -1 \\ -2 \\ -3 \end{pmatrix} \in \mathscr{M}_{3,1}.$$

Vérifiez que vos matrices ainsi définies sont du bon format notamment le format de la matrice devrait être un tuple de deux entiers.

[Q7.] Construire une fonction `alterne(n)` qui prend en entrée un entier `n` et renvoie en sortie la matrice $L \in \mathscr{M}_{1,2n}$ définie par :

$$L = (\underbrace{1\ 0}_{n \text{ fois}} \ldots 1\ 0)$$

[Q8.] Que font les commandes suivantes ?

```
1 def c(i,j):
2     return i**j
3
4 n,p = 3,3
5 C = np.array([[ c(i,j) for j in range(p)] for i in range(n)])
```

[Q9.] La matrice de Hilbert est la matrice $H \in \mathscr{M}_n$ dont le coefficient général est $h_{i,j} = \dfrac{1}{i+j-1}$. S'inspirer de l'exemple précédent pour construire une fonction `Hilbert(n)` qui fait ce que vous pensez.

[Q10.] Taper dans la console : `A`, `A[1,:]`, `A[:,2]`. Que font ces deux dernières instructions ?

[Q11.] Si on pose

$$D = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix},$$

construire cette matrice sous `Python`, puis remplacer dans D la ligne 0 et la colonne 3 de sorte à ce que D devienne : $\begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 \end{pmatrix}$

[Q12.] Tester dans la console les blocs de commandes suivants l'un après l'autre et dire ce que ces commandes font (n'hésitez pas à faire des affichages par `print` si besoin) :

```
Z = np.zeros((2,2))

U = np.ones([4,2])
U.T

I = np.eye(3)

Z2 = np.zeros_like(A)
Z2 = np.ones_like(A)
```

II Produit matriciel

II – A Programmation

[Q13.] **a)** Rappeler la formule du produit matriciel

b) Programmer une fonction `promat(A,B)` prenant en entrée deux matrices A, B de format compatible et renvoyant en sortie la matrice AB.

c) Vérifier que la programmation est correcte en essayant avec la fonction `np.dot` de `numpy`.

II – B Puissances successives

[Q14.] Programmer une fonction `puissance(A,n)` prenant en entrée une matrice carrée A, un entier `n` et renvoyant en sortie la matrice A^n. On utilisera au choix la fonction `np.dot` ou la fonction `promat`

III Exercices

III – A Test d'appartenance

[Q15.] Soit $\mathscr{E} = \left\{ \begin{pmatrix} a & a+b \\ b & a-2b \end{pmatrix} \quad (a,b) \in \mathbf{R}^2 \right\}$

Écrire une fonction `estDansE(M)` qui prend en entrée une matrice `M` de $\mathscr{M}_2(\mathbf{R})$ et qui renvoie `True` si M est dans $\mathscr{E}$ et `False` sinon.

III – B Changement de base

[Q16.] Calculer l'inverse de la matrice $J = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ avec la commande `la.inv(J)`.

On considère le sous-espace vectoriel $\mathscr{H}$ de $E = \mathbf{R}_4[X]$ des polynômes admettant 1 comme racine. Un polynôme sera représenté par la matrice de ses coordonnés sur la base canonique de E.

[Q17.] Écrire une fonction `estDansH(P)` prenant en entrée un polynôme P et renvoyant `True` si P est dns E et `False` sinon.

[Q18.] **a)** Donner une base $\mathscr{B}$ de $\mathscr{H}$.

b) Écrire une fonction `coord(P)` prenant en entrée un polynôme P de E et renvoyant en sortie :

1) La matrice des coordonnées de P sur la base $\mathscr{B}$ si P est dans E.

2) Un message d'erreur sous forme de chaine de caractères si P n'est pas dans E.

On pourra tester sa fonction sur le polynôme $P = 2X^4 - 9X^3 + 17X^2 - 14X + 4$.

Thème 10 : Méthodes de tri

Le but de ce TP est de programmer des fonctions permetttant, pour une liste de flottants `L` donnée, de renvoyer la liste triée dans l'ordre croissant des éléments de `L`.

I Modification d'un objet structuré par effet de bord

Tester les deux fonctions suivantes et comparer :

Sans effets de bord

```
def touchepas(L):
    M=L[:]
    M[0]=-15
    return M
```

```
>>> L=[1,2,3]
>>> L2 = touchepas(L)
>>> L[0],L2[0]
```

Avec effets de bord

```
def touche(L):
    L[0]=-15
```

```
L=[1,2,3]
>>> touche(L)
>>> L[0]
```

II Méthodes de secours

II – A Tri artisanal

a) Programmer une fonction `pos_min(L)` prenant en entrée une liste `L` de flottants et renvoyant en sortie l'indice dans `L` de son élément minimal.

b) En déduire une fonction `tri_basique(L)` prenant en entrée une liste `L` et renvoyant en sortie la liste `M` obtenue en rangeant les éléments de `L` dans l'ordre croissant.

II – B Tri artisanal bis

Pour ce tri, on suppose, quitte à d'abord calculer le maximum des éléments de la liste à trier, que les éléments de la liste à trier sont à valeurs dans $\{1 \dots N\}$, *N étant une constante connue.*

[Q1.] **a)** Écrire une fonction `multi(L,a)` prenant en entrée une liste `L`, un flottant `a` et renvoyant en sortie le nombre d'occurrences de `a` dans `L`. Par exemple, `multi([1,2,1,3,4,1,2],1)` de-

vrait renvoyer 3, et d'occurrences de `a` dans `L`. Par exemple, `multi([1,2,1,3,4,1,2],6)` devrait renvoyer `0`.

b) Déduire de cette dernière fonction une autre fonction `tri_bis(L)` de tri de liste de flottants.

III Tri à bulles (bubblesort)

Principe. Il repose sur le fait qu'une suite finie $(u_j)_{0\leq j\leq n-1}$ est triée si et seulement si :

$$\forall j \in \{0,\dots,n-2\} \qquad (P_j) : \quad u_j \leq u_{j+1}$$

Algorithme.

1. On effectue un parcours des éléments de la liste `L`, et chaque fois que l'on trouve lors de ce parcours deux termes consécutifs `L[j]`, `L[j+1]` ne vérifiant pas la propriété (P_j), on les permute. Cette dernière sera réalisée avec la fonction auxiliaire `permute(L,j)`.
2. Au bout du premier parcours, le plus grand élément de la liste est nécessairement le dernier. On en conclut deux choses :

   1) Si la liste est de longueur n, au bout d'un passage, on se retrouve à trier une liste de longueur $n-1$ à l'étape suivante.

   2) parcours suivant, il est inutile d'examiner la dernière paire puisque cette dernière n'aura pas à être permutée.

[Q2.] Écrire une fonction `permute(L,i,j)` prenant en entrée une liste flottants `L`, deux entier `i`,`j` et qui renvoie en sortie la liste `M` obtenue en échangeant les positions des éléments `L[i]` et `L[j]`.

[Q3.] En déduire une fonction `bubbleosrt(L)` prenant en entrée une liste de flottants `L` et renvoyant en sortie la liste `M` des termes de `L` triés dans l'ordre croissant.

[Q4.] Même question que la précédente, mais cette fois, la fonction agit directement sur `L` par *effet de bord* : elle ne renvoie aucun objet en sortie, mais modifie `L`.

IV Tri insertion

C'est la méthode la plus intuitive de tri, puisque c'est celle que l'on utilise naturellement par exemple pour trier ses cartes dans une partie de belote ou de tarot.

Algorithme.

a) On suppose qu'à chaque étape de l'algorithme, les éléments situés à gauche d'un élément privilégié de la liste, appelé **pivot** de la liste sont correctement triés. C'est ce pivot qui sera à insérer correctement pour réaliser le tri.

b) À la première étape, le pivot p est le premier élément de la liste.

c) À chaque étape, on compare le pivot p à l'élément z qui le **précède** dans la liste. correctement :

— Si $p < z$, c'est que p est mal placé, donc on le rétrograde dans la liste (à gauche donc) jusqu'à ce qu'il soit en bonne position dans la liste.

— Sinon c'est que p est bien placé.

d) On répète `3.` (ce qui constitue une étape) avec le terme w qui était à droite de p à l'étape précédente : w devient le nouveau pivot.

e) Comme la position du pivot augmente strictement à chaque étape, à la fin, tous les éléments auront été des pivots, et la liste est triée.

[Q5.] Écrire une fonction `retrograde(L,i)` qui prend en entrée une liste `L` de flottants, un entier `i`, et qui renvoie en sortie la liste `M` obtenue en rétrogadant `L[i]` de sorte que le premier élément précédent `L[i]` dans `M` luis soit strictement inférieur.

[Q6.] En déduire une fonction `tri_insertion(L)` prenant en entrée une liste de flottants `L` et renvoyant en sortie la liste `M` des termes de `L` triés dans l'ordre croissant.

[Q7.] Même question que la précédente, mais cette fois, la fonction agit directement sur `L` par *effet de bord* : elle ne renvoie aucun objet en sortie, mais modifie `L`.

V Tri rapide (quicksort)

C'est un tri récursif, par définition. Son intérêt est qu'il est plus rapide (en termes de nombre d'opérations) que les autres méthodes de tri qui ont été programmées. En voici une implémentation :

```
1  def qs(L):
2      n  = len(L)
3      if n<=1:
4          return L
5      else:
6          pivot = L[0]
7          Lgauche =[]
8          Ldroite =[]
9          for i in range(1,n):
10             if L[i] < pivot:
11                 Lgauche.append(L[i])
12             else:
13                 Ldroite.append(L[i])
14     return qs(Lgauche)+[pivot]+qs(Ldroite)
```

[Q8.] Expliquer son principe.

VI Applications

VI – A Médiane d'une statistique univariée

Définition. Soit $x = (x_1 < \cdots < x_n)$ une série statistique de longueur n (triée). La médiane Q_2 de la série est :

$$Q_2 = \begin{cases} x_{1+\frac{n-1}{2}} & \text{si n est impair} \\ \frac{1}{2}\left(x_{\frac{n}{2}} + x_{1+\frac{n}{2}}\right) & \text{si n est pair} \end{cases}$$

[Q9.] Programmer une fonction `mediane(L)` prenant en entrée une liste de flottants `L` représentant une série statistique et donnant en sortie la valeur de sa médiane (attention à la numérotation des éléments d'une liste en python)

VI – B Tri d'une liste en fonction d'une autre

[Q10.] Écrire une fonction `tri_couples(L)` prenant en entrée une liste `L` de 2-tuples (a, b) tels que a est un flottant, et renvoyant en sortie la liste `M` obtenue en rangeant les éléments de `L` par ordre croissant des a.

[Q11.] En déduire une fonction `tri2(L1,L2)` prenant en entrée deux listes de même longueur `L1`, `L2`, la liste `L1` étant une liste de flottants, et renvoyant en sortie la liste `L2` triée par ordre croissant des éléments de `L1`.

Thème 11 : Diagonalisation des matrices carrées

```python
import numpy as np
import numpy.linalg as la
import random as rd
```

I Préliminaires

La commande `la.matrix_rank(M)` donne le rang d'une matrice M.

[Q1.] Écrire une fonction `dimKer(M)` prenant en entrée une matrice carrée M et renvoyant en sortie la dimension du sev $\text{Ker}(M)$.

[Q2.] Écrire une fonction `dimE(A,mu)` prenant en entrée une matrice carrée A, un flottant μ et renvoyant en sortie la dimension du sous-espace $\text{Ker}(A-\mu I_n)$, n étant le nombre de lignes de A.

[Q3.] En déduire une fonction `est_diago(A,L)` prenant en entrée une matrice carrée A, la liste `L` de ses valeurs propres, et renvoyant en sortie `True` si A est diagonalisable et `False` sinon.

Dans le formulaire python du concours Agro-Veto fourni à l'oral jusqu'en 2023, on peut lire ceci :

« *`la.eig(M)` renvoie une couple `L,P` où `L` est la liste des valeurs propres de M et P la matrice de passage associée.* »

[Q4.] Cette formulation du concours se veut brève pour le formulaire, mais est abusive mathématiquement. Expliquer pourquoi.

II Diagonlisation

[Q5.] La liste `L` obtenue avec `la.eig(M)` répète les valeurs propres de `M`. Écrire une fonction `spectre(M)` qui prend en entrée une matrice M et qui renvoie en sortie la liste de ses valeurs propres **distinctes**.

[Q6.] Soit $J = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$. À la main : donner le spectre de J. La matrice J est-elle diagonalisable ?

[Q7.] Pour chacune des matrices A suivantes, et à l'aide de l'ordinateur et des fonctions précédentes :

a) calculer son spectre,

b) dire si A est diagonalisable,

c) donner une base de chaque sous-espace propre, et le cas échéant, une matrice P inversible diagonalisant A.

```
 ============================================================
[[ 0  1  0]
 [ 0  0  1]
 [ 0  0  0]]
============================================================
[[-5   6   4]
 [-4   5   4]
 [ 2  -2  -3]]
============================================================
[[ 1   3   2]
 [ 0  -2  -2]
 [-3   9   8]]
============================================================
[[ 3   5  -3]
 [ 0   2   0]
 [ 6  10  -6]]
 ============================================================
[[ 4  -2   0]
 [ 4  -2   0]
 [-2   2   2]]
============================================================
[[ 0   1   0]
 [-2   3   0]
 [ 0   0   2]]
============================================================
[[ 1   4   2]
 [ 2   3   2]
 [-4  -8  -5]]
============================================================
[[ 4   1   2]
 [-1   1  -1]
 [-2  -1   0]]
============================================================
[[ -1    5   -5]
 [  1   -7    6]
 [  1  -11   10]]
============================================================
[[ 1  -1   0]
 [ 1   3   0]
 [-1  -3   1]]
============================================================
```

Thème 12 : Simulation de variables aléatoires discrètes

```python
import numpy as np
import random as rd
```

I Simulation d'expériences aléatoires

Rappel de quelques notions sans rentrer dans des définitions mathématiques :

a) Une expérience est un processus **reproductible** conduisant à un résultat **observable**.

b) Une expérience est dite aléatoire si en la reproduisant, le résultat observé n'est pas toujours le même. Sinon, elle est dite déterministe.

c) Une variable aléatoire est une fonction qui associe à tout résultat d'une expérience aléatoire donnée un nombre. Typiquement : une mesure sur l'observation faite à l'issue d'une expérience aléatoire.

d) Un ordinateur ne fabrique rien d'aléatoire. En fait, python possède une suite récurrente prédéfinie (dont la formule la définissant est parfaitement déterministe) dont le comportement des termes est si imprévisible qu'on peut qualifier son comportement d'aléatoire.

I – A Simulation d'expériences aléatoires

— La commande `rd.randint(a,b)` renvoie, pour deux entiers $a \leq b$ un entier choisi au hasard dans l'intervalle $[a, b]$.

— En revanche la commande `rd.randrange(a,b)` renvoie au hasard un entier choisi au hasard dans `range(a,b)`.

— La commande `rd.choice(L)` choisit un élément au hasard dans une liste `L` (rem : en python, les éléments d'une liste s'appellent ses items).

Ces fonctions permettent déjà simuler de nombreuses expériences aléatoires.

[Q1.] Simuler les expériences suivantes :

a) On lance une pièce et on regarde sur quel côté la pièce est tombée. Estimer ensuite empiriquement la probabilité de pile c'est-à-dire calculer la informatiquement sur un grand nombre de simulations la fréquence observée du pile.

b) On lance un dé. On regarde le score. Estimer ensuite empiriquement la probabilité d'obtenir 2.

c) On lance deux dés. On regarde le total des points affichés. Estimer ensuite empiriquement la probabilité d'obtenir 8.

d) Une urne contient 10 boules numérotées de 1 à 10. On tire une boule au hasard et on regarde le numéro obtenu. Estimer ensuite empiriquement la probabilité d'obtenir le numéro 1.

e) Même urne que précédemment, mais on tire 5 fois de suite avec remise, et on observe la suite des numéros observés. Estimer ensuite empiriquement la probabilité d'obtenir un 10 dans la suite.

f) Même urne que précédemment, mais on tire 5 fois de suite avec remise, et on compte le nombre de 3 observés dans la série des 5 tirages. Estimer ensuite empiriquement la probabilité d'obtenir exactement deux occurrences du «3» dans la suite.

g) Même urne que précédemment, mais on tire avec remise jusqu'à ce que l'on tire le 10. On observe le nombre de tirages nécessaires pour y arriver. Estimer ensuite empiriquement la probabilité de faire au moins 5 tirages pour observer le 10.

h) Même urne que précédemment, mais on tire 5 fois de suite sans remise et on observe la suite des numéros observés. Estimer empiriquement la probabilité que le numéro 10 figure dans la suite.

i) Même urne et même protocole que précédemment. Estimer empiriquement la probabilité que les résultats des tirages forme une suite croissante.

[Q2.] Soit $c \in \mathbf{N}^*$. Une urne contient 2 boules blanches et 2 boules noires, on y effectue un premier tirage, puis on suit le protocole suivant :

Si la boule obtenue est noire, on la remet dans l'urne, on ajoute c boules noires, et on effectue un nouveau tirage,

si la boule obtenue est blanche, on s'arrête. Ecrire une fonction prenant en argument deux entiers naturels non nuls c et s et qui renvoie une valeur approchée de la probabilité de l'évènement : «au bout de s tirages, on n'a pas observé de blanc.»

[Q3.] Soit $n > 2$ un entier. Dans une assemblée de n brebis, tout le monde lance simultanément une pièce. Si toutes les pièces tombent du même côté sauf une, la brebis propriétaire de cette dernière est déclarée brebis galeuse. Simuler une réalisation de cette expérience renvoyant le nombre de lancers nécessaires à l'apparition d'une brebis galeuse.

I – B Variable aléatoire de Bernoulli

La commande `rd.random()` tire au hasard un flottant dans l'intervalle $[0,1[$.

[Q4.] **a)** Dessiner le segment $S = [0,1[$ et représenter sur ce segment le style de nuage de points que l'on devrait obtenir si on un nombre important de flottants à l'aide de `rd.random()`.

b) Soit p un réel fixé donné dans $]0,1[$ (par exemple $1/2$ ou $1/3$, ou $3/4\ldots$). Tracer p sur S. Quelle proportion (environ) de points du nuage tracé précédemment se situent à gauche de p ?

c) En déduire une fonction `Bernoulli(p)` prenant en entrée un flottant p et simulant le comportement d'une variable aléatoire de Bernoulli de paramètre p.

[Q5.] Soit $p \in]0;1[$. Une urne contient des boules noires en proportion p et des boules blanches en proportion $1-p=q$. On effectue dans cette urne des tirages successifs avec remise.

Soit X la variable aléatoire égale au nombre de tirages juste nécessaires pour obtenir les deux couleurs.

Écrire une fonction `X(p)` permettant de simuler la variable aléatoire X.

I – C Variable binomiale

a) On rappelle qu'un schéma de Bernoulli de paramètres (N,p) est une expérience aléatoire consistant en N répétitions mutuellement indépendantes d'une même épreuve de Bernoulli de paramètre p.

b) On rappelle qu'une variable aléatoire X suit une loi $\mathscr{B}(N,p)$ si elle compte le nombre de succès d'un schéma de Bernoulli de paramètres (N,p).

[Q6.] Écrire une fonction `binomial(N,p)` simulant une variable aléatoire de loi $\mathscr{B}(N,p)$.

[Q7.] Une urne contient 5 boules blanches et 3 boules noires. On réalise l'expérience suivante : « On tire 30 fois avec remise une boule de cette urne. » Écrire une fonction en python `nblanhes()` ne prenant aucune variable en entrée et renvoyant en sortie le nombre de boules blanches obtenues par simulation de cette expérience.

I – D Variable géométrique

On lance une pièce de monnaie équilibrée. On relance la pièce jusqu'à obtenir pile. On note X la variable aléatoire égale au nombre de lancers qui ont dû être effectué pour observer ce pile.

[Q8.] Programmer une fonction `G()` ne prenant aucune variable en entrée et simulant la variable X.

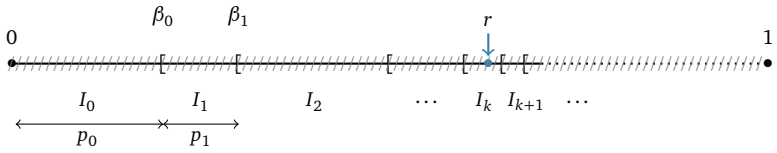
I – E Simulation d'une VAR discrète quelconque

Soit X une VAR sur un espace probabilisé, notons $X(\Omega) = \{x_0 < x_1 < \cdots < x_{n-1} < \ldots\}$ son espace image (au plus dénombrable) et notons $p_k = P(X = x_k)$.

a) Comme la somme $\sum_{k\geq 0} p_k$ vaut 1, on peut partitionner l'intervalle $[0,1[$ en intervalles de I_k de longueur p_k comme sur la figure.

b) On tire un flottant r au hasard dans $[0,1[$ (on a représenté sur le dessin un grand nombre de tirages de ces flottants).

c) *Simulation de X* : comme la proportion de flottants r tombant dans I_k sur un grand nombre de tirages (déssinés en hachures) est environ p_k, on décrète que si r tombe dans I_k, la valeur observée de X est alors x_k (puisque x_k est alors observé en moyenne avec la fréquence p_k).



[Q9.] Compléter la phrase : soit r un réel dans $[0,1[$ comme sur la figure. L'unique entier k tel que $r \in I_k$ est le plus .....entier k tel que r ..... β_k.

Soit r un réel dans [0,1[comme sur la figure. L'unique entier k tel que $r \in I_k$ est le plus petit entier k tel que $r < \beta_k$.

[Q10.] Soit k un entier. Que vaut β_k en fonction des nombres p_j ?

Par construction des intervalles I_k, $\beta_k = \sum_{j=0}^{k} p_j$

[Q11.] Écrire une fonction `sumcum(L)` qui prend en entrée une liste de flottants `L` et qui renvoie la liste des sommes cumulées des items de `L`, c'est-à-dire la liste des termes $s_0, \ldots, s_{n-1}$ où $s_k = \sum_{j=0}^{k} \texttt{L[j]}$.

[Q12.] Én déduire une fonction `simul(X,P)` qui prend en entrée deux listes de flottants `X=[x[0]...x[n-1]]` et `p=[p[0]...p[n-1]]` représentant respectivement la liste $x_0 < x_1 < \cdots < x_{n-1}$ des n valeurs de l'espace image d'une variable aléatoire X donnée, ainsi que la liste des probabilités $P(X = x_k)$ correspondantes, et qui renvoie en sortie la valeur x_k avec la probabilité p_k suivant le principe rappelé précédemment. Cette fonction simule bien la variable aléatoire X.

[Q13.] Soit X une variable aléatoire dont la loi est la suivante :

$k=$	0	1	2
$P(X=k)=$	1/2	1/3	1/6

Simuler cette variable aléatoire informatiquement.

[Q14.] C'est l'anniversaire de la petite chienne Cybèle, et ses amis humains viennent lui présenter leurs voeux. On suppose que le nombre N de personnes ayant fait le déplacement pour cet évènement est variable aléatoire de loi de Poisson de paramètre 15. Pour remercier ses fans, Cybèle mord ses visiteurs avec une probabilité $p = 1/3$, les morsures étant supposées mutuellement indépendantes.

Soit X la variable aléatoire égale au nombres d'invités mordus. Simuler la variable X.

Thème 13 : Simulation de variables aléatoires à densité

```
1 import random as rd
2 import numpy as np
3 import matplotlib.pyplot as plt
```

Il est souhaitable de disposer du formulaire python du concours Agro-Veto, et de papier pour calculer !

I Lois usuelles

[Q1.] **a)** Écrire une fonction `Unif(a,b)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité d'une loi uniforme sur le segment $[a, b]$.

b) Écrire une fonction `expo(mu)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité de loi exponentielle de paramètre μ.

c) Écrire une fonction `normale(mu,sigma)` dont l'exécution renvoie en sortie une simulation d'une variable aléatoire à densité de loi normale de paramètres μ, σ^2. *Utiliser le formulaire*

Le script suivant permet de représenter l'histogramme des fréquences d'une loi normale centrée réduite, ce qui donne l'allure d'une densité :

```
1 N = 100000                              # nombre de simulations
2 n = 50                                  # nombre de rectangles dans l'
    histogramme
3 L=[rd.gauss(0,1) for _ in range(N)] # série de N simulations
4 plt.hist(L,n,normed=True)               # normed = True : pour un
    histogramme de
5 plt.show()                              # fréquences (somme des aires des
    bâtons = 1)
```

[Q2.] Reproduire ce script, et constater qu'on a bien l'allure des densités de `[Q1.]` conformes à celles du cours.

Remarque. Si vous avez un message d'avertissement en exécutant ce script, remplacez le nom d'option `normed` en ligne 4 par `density`.

II Exercices

[Q3.] **a)** Si X suit une loi exponentielle de paramètre $\lambda > 0$ et Y suit une loi exponentielle de paramètre $\mu > 0$, simuler la variable $Z = \min(X, Y)$.

b) Calculer la loi de Z mathématiquement.

c) Montrer que la loi de Z est une loi exponentielle de paramètre α à préciser.

d) Constater en traçant les histogrammes des fréquences pour la variable Z et une variable de loi $\mathscr{E}(\alpha)$ la correction de votre réponse à `3.`

[Q4.] Soit $\lambda > 0$ un réel, et f la fonction définie par : $f(x) = \begin{vmatrix} 0 & \text{si} & x \leq 1 \\ \dfrac{\lambda}{x\sqrt{x}} & \text{si} & x > 1. \end{vmatrix}$

a) Déterminer λ tel que f soit une densité de probabilité d'une variable aléatoire X.

b) Déterminer la loi de la variable alétaoire $Y = \ln X$.

c) En déduire une méthode de simulation de la variable Y.

[Q5.] Soit $a > 0$ un réel et f_a la fonction définie sur $\mathbf{R}$ par $f_a(x) = \dfrac{a}{\pi(x^2 + a^2)}$.

a) Montrer que f_a est une densité.

b) Montrer que si U suit une loi uniforme sur $]0, 1[$, alors la variable $Z = \tan\left(\pi U + \dfrac{\pi}{2}\right)$ est une variable à densité de densité f_1.

c) Simuler une variable aléatoire X de loi f_1.

d) Estimer informatiquement plusieurs fois l'espérance de la variable X. La variable X admet-elle une espérance ?

e) Soit $a > 0$. Simuler une variable de loi f_a.

[Q6.] **a)** Soit $a \in \mathbf{R}$, $b > 0$ et $f_{a,b}$ la fonction définie sur $\mathbf{R}$ par :

$$f_{a,b}(x) = \frac{1}{2b} \exp\left(-\frac{|x-a|}{b}\right).$$

1) Montrer que $f_{a,b}$ est une densité.

2) Soit U une variable aléatoire de loi exponentielle de paramètre 1. Soit V une variable aléatoire de loi uniforme sur l'ensemble fini $\{-1, 1\}$. On suppose que U et V sont indépendantes. Montrer que la variable $X = a + bUV$ est à densité et admet pour loi la densité $f_{a,b}$.

3) Simuler la variable X de la question précédente.

4) Estimer empiriquement l'espérance et la variance de X.

Thème 14 : Statistiques

```python
import numpy as np
import matplotlib.pyplot as plt
import random as rd
from scipy.stats import norm
```

I Autour de la loi normale centrée réduite

[Q1.] **a)** Tracer la courbe de la densité $\mathcal{N}(0,1)$ sur l'intervalle $[-3,3]$.

b) Pourquoi avoir choisi cet intervalle ?

c) Tracer la courbe de la fonction de répartition Φ de la densité $\mathcal{N}(0,1)$ en la programmant avec une méthode des rectangles sur un segment bien choisi à l'aide de la question précédente.

d) Comparer le tracé avec celui de la fonction implémentée par défaut sous python :

```python
Phi = norm.cdf
```

II Autour du théorème central-limite

II – A Quelques définitions

Définition 1 Si X est une variable aléatoire réelle définie sur un espace probabilisé $(\Omega, \mathcal{T}, P)$, et n un entier strictement positif, un **n-échantillon de X** (ou de la loi de X) est une famille de n variables aléatoires mutuellement indépendantes définies sur Ω et de même loi que X.

Définition 2 La **moyenne empirique de X** (ou de la loi de X) est la suite $(M_n)_{n\geq 1}$ de variables aléatoires définies par :

$$\forall n \geq 1 \quad M_n = \frac{X_1 + \cdots + X_n}{n}$$

où $(X_1, \ldots, X_n)$ est un n-échantillon de X.

Définition 3 Un ***run*** de longueur n de X est une n-liste de réels $(x_1, \ldots, x_n)$ telle que pour tout entier $i \in \{1, \ldots, n\}$, x_i est une réalisation de X_i, les X_i constituant un n-échantillon de la loi de X. Par exemple, une suite 50 lancers de dés non pipés mutuellement indépendants est un run de longueur 50 d'une loi $\mathcal{U}(6)$.

II – B Théorème de Moivre-Laplace

[Q2.] **a)** Écrire une fonction `B(p)` prenant en entrée un flottant $p \in]0;1[$ simulant une variable aléatoire de Bernoulli de paramètre p.

b) Écrire une fonction `Mn(n,p)` prenant en entrée un entier n, un flottant p et renvoyant en sortie une simulation de la moyenne empirique M_n d'une loi $\mathscr{B}(p)$.

c) Créer sous forme de liste un run de longueur `N` de la variable M_{30}, avec par exemple $p = 1/2$. On pourra prendre `N=100`.

d) En déduire un run de longueur `N`, sous forme de liste également de la variable $M_{30}^{\star}$ (Variable centrée réduite de M_{30}). On peut par exemple fixer `N=1000`.

e) Tracer sur un même graphique l'histogramme des valeurs observées de $M_{30}^{\star}$ et de la densité $\mathscr{N}(0,1)$.

f) Faire varier la taille n de l'échantillon ainsi que la longueur `N` du run des observations.

Remarque. On rappelle que la commande pour tracer l'histogramme des fréquences d'une série de données `L` :

```
nb = 100 # nombre de bâtons dans l'histogramme
plt.hist(L,nb,normed=True,ec='w',alpha=0.8)
```

III Convergence en loi

La notion permettant d'évaluer la proximité de deux lois est celle de convergence en loi. Cette dernière se définit assez simplement à l'aide des fonctions de répartition (voir le cours sur les théorèmes limites).

La commande pour tracer le polygone des fréquences cumulées d'une série d'observations `L` est la suivante :

```
plt.hist(L,nb,cumulative=True, density=True,histtype='step')
```

[Q3.] Tracer sur un même graphique la fonction de répartition de la loi $\mathscr{N}(0,1)$ et le polygone des fréquences cumulées du run centré réduit pour la loi de Bernoulli.

III – A Expérimentations

Le résultat de convergence en loi de $M_n^{\star}$ vers une loi $\mathscr{N}(0,1)$ n'est pas spécifique à la loi de Bernoulli, mais constitue un cas particulier du théorème central limite.

[Q4.] Observer ce phénomène de convergence en loi pour les lois suivantes :

a) $\mathscr{U}(6)$.

b) $\mathscr{E}(1)$.

[Q5.] Proposer une méthode de simulation de loi normale centrée-réduite.

Thème 15 : Graphes

```python
import numpy as np
```

I Notion de graphe

I – A Définition

Définition 1 : *Graphe à n sommets, sommets, arêtes* Soit $n \in \mathbf{N}^*$.

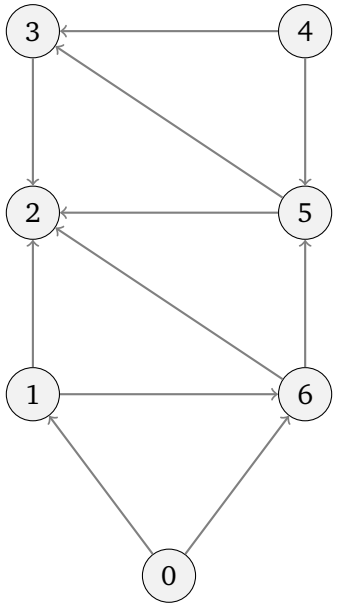
- Graphe orienté $G = (V, A)$: c'est la donnée :
  - d'un ensemble fini d'entiers $V = \{0, 1, \ldots, n-1\}$, appelés **sommets** du graphe,
  - et d'un ensemble A de **couples** d'éléments de V. Les éléments de A sont appelés **arêtes** du graphe.
- Graphe non orienté : même définition que celle de graphe orienté, à ceci près que l'ensemble A est un ensemble de *paires* d'éléments de V.

I – B Représentation graphique d'un graphe

En général :

- les sommets d'un graphe G sont représentés par des cellules (des pastilles, quoi),
- et les arêtes :
  - dans le cas de graphes orientés : par des flèches allant de i vers j si $(i, j) \in A$.
  - dans le cas de graphes non-orientés : simplement par un segment liant i à j si $\{i, j\} \in A$.

L'exemple représenté en page suivante est un exemple de représentation d'un graphe G qui servira de terrain d'application pour toute la suite du propos.



Graphe G de référence

[Q6.] Donner ci-après les ensembles A et V de ce graphe G après avoir dit si il est orienté ou non.

Définition 2 : *Voisin d'un sommet i d'un graphe orienté* Soit $G = (A, V)$ un graphe orienté et $i \in V$. On dit que le sommet $j \in V$ est un voisin de i si (i, j) est une arête de G, c'est-à-dire : $(i, j) \in A$.

[Q7.] Écrire mathématiquement la définition de voisin pour un graphe non orienté :

I – C Représentation matricielle d'un graphe

Définition 3 : *matrice d'adjacence* Soit $G = (A, V)$ un graphe, et notons $V = \{1, \ldots, n\}$ ses sommets. On appelle matrice d'adjacence de G la matrice $M \in \mathscr{M}_n(\mathbf{R})$ de coefficient $m_{i,j}$ égal à :

— 1 si les sommets i et j sont reliés par une arête.

— 0 sinon.

[Q8.] **a)** Quelle propriété remarquable possède la matrice d'adjacence d'un graphe non orienté ?

b) Écrire la matrice d'adjacence du graphe G de l'exemple 1, et la matrice d'adjacence de la version non orientée de ce même graphe.

[Q9.] Soit $G = (A, V)$ un graphe orienté de matrice d'adjacence de G la matrice M. En termes de voisins :

a) Que représente la somme des éléments de la ligne i dans M ?

b) Que représente la somme des éléments de la colonne j dans M ?

I – D Implémentations possibles sous Python

Comme toujours en informatique, l'implémentation efficace d'un objet repose sur un choix adapté de la structure de données pour le représenter et résoudre le problème souhaité. Sous **Python**, on peut représenter un graphe :

a) par sa martrice d'adjacence.

b) par une simple liste `L`, appelée **Liste d'adjacence** du graphe. Dans ce cas pour $j \in V$, `L[j]` contient la liste (éventuellement vide) des voisins du sommet `j` dans le graphe. La liste `L` est donc de longueur Card V.

[Q10.] Définir en Python la liste d'adjacence `L_adj` du graphe de l'exemple 1.

[Q11.] **a)** Écrire une fonction `L2M(M)` qui prend en entrée une matrice d'adjacence d'un graphe et qui renvoie en sortie sa liste d'adjacence.

b) Écrire une fonction `M2L(L)` qui prend en entrée la liste d'adjacence d'un graphe et qui renvoie en sortie sa matrice d'adjacence.

I – E Chemin dans un graphe

Défintion 4 :*Chemin dans un graphe, cycle d'un graphe, longueur d'un chemin*

a) Un chemin dans un graphe G entre deux sommets i et j de ce graphe est une suite finie $(s_0, \dots, s_\ell)$ de sommets du graphe telle que :

— $s_0 = i$,

— $s_\ell = j$,

— Pour tout entier k dans $\{1, \dots, \ell\}$, s_k est un voisin de s_{k-1}.

b) Avec les notations précédentes, un chemin pour lequel $s_0 = s_\ell$ et $s_k \neq s_0$ sinon s'appelle un cycle du graphe.

c) La longueur du chemin $(s_0, \dots, s_\ell)$ est par définition égale à ℓ.

Exemple.

— Une arête est un chemin de longueur 1.

— Dans le graphe de l'exemple 1 :

— $(0, 1, 6, 2)$ est un chemin de longueur 3 allant de 0 à 2.

— $(2, 5, 3)$ n'est pas un chemin de 2 à 3.

[Q12.] Soit $G = (A, V)$ un graphe de matrice d'adjacence la matrice $M \in \mathcal{M}_n(\mathbf{R})$.

a) Écrire la formule du produit matriciel pour le coefficient $d_{i,j}$ de la matrice M^2 et soit $m_{i,k} \times m_{k,j}$ un terme donné de la somme définissant $d_{i,j}$

b) Soit k un entier fixé Quelles sont les valeurs possibles du produit $m_{i,k} \times m_{k,j}$, et de quoi mesure-t-il l'existence en termes de chemins dans G ?

c) En déduire ce que représente le coefficient $d_{i,j}$ dans M^2 en termes de chemins dans G.

d) De façon générale (pas de preuve attendue) :

1) Que représente le coefficient en position (i, j) dans M^k ?

2) Que représente la somme des coefficients diagonaux de M^k ?

[Q13.] **a)** Écrire une fonction `nbcycles(M,l)` prenant en entrée une matrice d'adjacence `M` d'un graphe G, un entier `l` et renvoyant en sortie le nombre de cycles de longueur `l` dans G

b) Calculer le nombre de cycles de ℓ ($\ell \in \{2, \ldots, 5\}$) dans le graphe G de l'exemple 1.

II Parcours en largeur (algorithme BFS)

— Dans la plupart des situations qui nous intéressent, les graphes étudiés sont gigantesques, il est hors de question, voire impossible, d'entrer la matrice ou la liste d'adjacence dans la machine.

— Néanmoins, pour ces mêmes situations, le graphe en lui-même ne nous intéresse pas, c'est plutôt l'existence d'un chemin reliant un sommet donné s à un autre qui importe, précisément le parcours effectué pour aller de s à un sommet x d'intérêt.

— L'idée est alors de découvrir le graphe par une exploration systématique (mais pas forcément exhaustive !) à partir de s. L'algorithme de *parcours en largeur* [1] est un type de parcours systématique. Son implémentation est un incontournable de l'informatique, car il donne de façon inattendue une solution optimale à des problèmes courants, c'est pourquoi il est présenté ici.

II – A Principe et implémentation

Le principe du parcours en largeur d'un graphe en partant d'un sommet s donné du graphe est le suivant.

1. En anglais, l'algorithme de parcours en largeur se dit : Breadth-first-seach algorithm (abrégé BFS).

Principe de l'algorithme Parcours en largeur depuis un sommet s.

a) On part de s.

b) On découvre tous les voisins de s (déf. 2) : ces voisins sont marqués comme découverts lors du parcours à partir de s, et insérés dans une «file d'attente» `File` l'un après l'autre. Cette file est la file des voisins qu'il reste à explorer. Pour systématiser l'exploration (et donc n'oublier personne !), on traite la file d'attente comme dans la vie : le premier arrivé dans la file est le premier à être examiné.

c) Ensuite, pour chaque voisin v de s qui vient d'être découvert lors du parcours (c'est-à-dire inséré dans la file), on répète les étapes précédentes `1.` et `2.` en prenant à cette étape v au lieu de s.

d) Si on ne découvre plus de voisins à v dans l'étape `2.`, cela signifie que la file est vide : le parcours est terminé. Il a révélé tous les sommets accessibles en largeur depuis s. Ce qui répond à la question de savoir si un sommet est accessible (par ce parcours) depuis s, et surtout, cela donne le chemin le plus court pour y arriver.

Implémentation en Python

```
1  def parcours(start):
2      """
3      Parcours en largeur depuis  start.
4      """
5      File = [(start,[start])] #(départ,
6                               # chemin)
7      vus = []
8      while File: # idem que
9                  # while len(File)>0:
10         s,path = File.pop(0)
11         if s not in vus:
12             vus.append(s)
13             V = Lvoisins(s)
14             for v in V:
15                 if not v in vus:
16                     File.append((v,path+[
                           v]))
17             if is_gagnant(s):
18                 return path
19     return []
```

Remarque Pour un graphe dont la liste d'adjacence n'est pas connue, le parcours en largeur reste possible : il suffit seulement de disposer d'une fonction capable de calculer les voisins d'un sommet donné du graphe pour chaque sommet découvert.

II – B Application

Afin de comprendre cet algorithme, on reprend le graphe de l'exemple 1, et on cherche à voir si le sommet 3 est accessible depuis 0 par ce parcours.

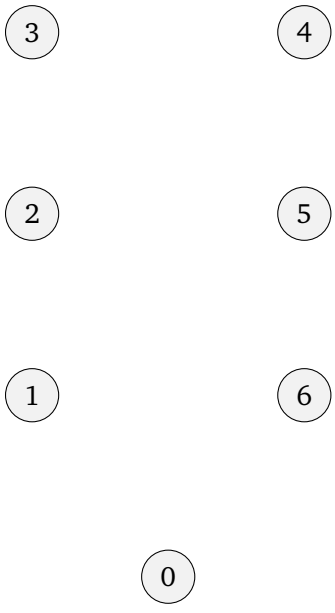
[Q14.] Programmer une fonction `Lvoisins(x,L)` qui prend en entrée un entier `x` représentant un sommet d'un graphe de liste d'adjacence `L`, et renvoyant en sortie la liste des voisins de `x` dans ce graphe.

[Q15.] Programmer une fonction `is_gagnant(x)` qui prend en entrée un entier `x` représentant un sommet d'un graphe et renvoyant en sortie le booléen `True` si et seulement si le sommet `x` vaut `3`.

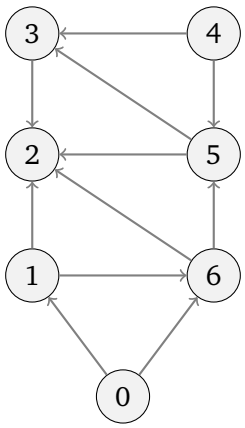
[Q16.] Reprendre le graphe de l'exemple 1 ci-contre, avec sa matrice d'adjacence `L_adj` et tester le parcours en largeur depuis le sommet 0 en exécutant le script `exemple-BFS.py`, puis en tapant dans la console :

```
>>> parcours(0,L)
```

Remplir le tableau ci-dessous au fur et à mesure du parcours, et matérialiser aussi au fur et à mesure les arêtes découvertes pour comprendre le fonctionnement de cet algorithme (il est bon d'avoir le graphe sous les yeux pour cela, il est reproduit ci-contre) :



Sommet `v` en cours	**État de la file** (↑ : sommet en traitement)	**Voisins découverts de `v`**
0	`[(0,[0])]` ↑	



[Q17.] Retester la fonction parcours, en considérant cette fois un parcours depuis le sommet `0` et en examinant si le sommet `4` est accessible par un parcours en largeur.