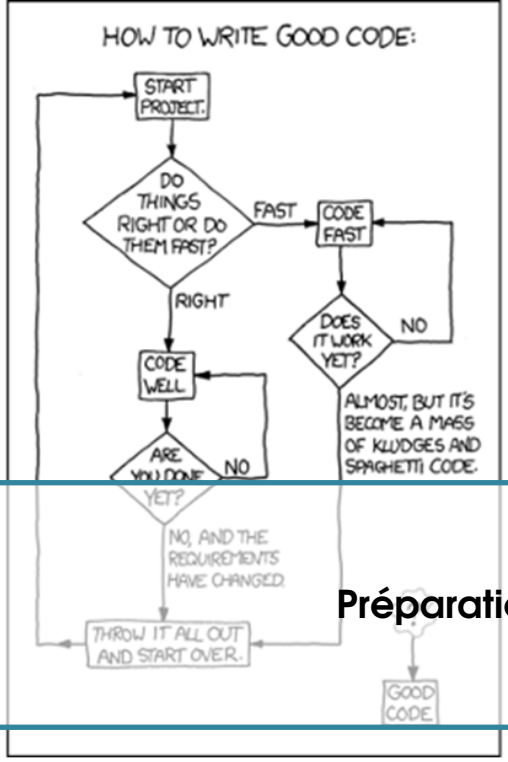
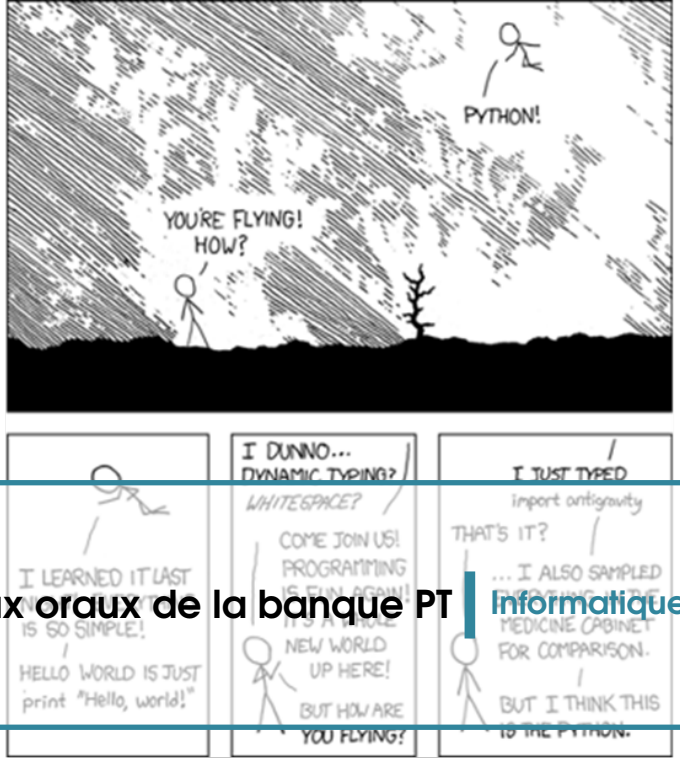




PT – PT ⋆

Oral

Préparation aux oraux de la banque PT | Informatique

Exercices

Préparation aux oraux de la banque PT
Épreuve de « Mathématiques et Algorithmique »

Xavier Pessoles

D'après Banque PT et L'informatique pas à pas en prépa.
PT – PT ⋆

1 Avant-propos

Ce recueil d'exercices est réalisé à partir de 3 sources :

- des exercices «zeros» de la banque PT ;
- d'exercices retranscrits par vos prédécesseurs en 2015 (Lycée Mimard de Saint-Étienne et Lycée La Martinière Monplaisir) ;
- d'exercices tirés du livre de Frédéric Butin : *L'informatique pas à pas en prépa. Cours et exercices corrigés. Éditions ellipses.*

Ces exercices ont pour but de vous entrainer à la partie «Informatique» de l'épreuve de «Mathématiques et d'algorithmique» de la banque PT, anciennement « Maths II». Cette épreuve se déroule à l'école Arts et Métiers ParitTech de Paris.

Afin de vous mettre dans les conditions de l'épreuve, je vous encourage très très très vivement à utiliser IDLE. IDLE est disponible avec toute installation de Python, Pyzo ou WinPython. Pour cela, aller dans le dossier contenant Pyzo puis dans le répertoire `Lib\idlelib` et lancer le programme `idle.bat`.

Pour réaliser les exercices vous aurez accès à un aide-mémoire des fonctions Python (qui devrait être agrafé à ce document si je n'oublie pas !

Notes pour les PT ⋆ (et les autres)

L'objectif pour vous est de réaliser le maximum d'exercices afin de vous entraîner afin que vous puissiez «parler python couramment». Vous aurez deux créneaux pour vous entrainer (ce qui ne vous dispense pas de vous entrainer chez vous). Un créneau en parallèle du TD de maths avec Mme Gaggioli. Pendant ce créneau vous travaillerez en autonomie. Un créneau le vendredi matin où je serai là pour vous aider ou pour répondre à vos questions.

Remarques concernant les corrigés

Je n'ai pas eu le temps de réaliser les corrigés de chacun des exercices. Concernant les corrigés des exercices de F. Butin, or mention spéciale, les corrigés sont ceux que je propose. Pour les exercices sans corrigé, on peut donc les retrouver dans le bouquin.

Pour les exercices retranscrits, il se peut qu'il y ait des erreurs de texte (de ma part ou de la part des anciens élèves). Merci de me les rapporter à l'adresse `xpessoles.ptsi@free.fr`.

Par ailleurs, vous pouvez proposer des corrigés que j'ajouterai à ce document ce qui vous vaudra d'avoir votre non cité dans ce recueil ainsi que mon entière considération !

Remarques concernant les exercices retranscrits

Comme vous le voyez, cette préparation repose en partie sur le retranscription des exercices de vos prédécesseurs. Afin d'améliorer la préparation des futurs élèves, je vous demande donc, à votre tour, à la fin de l'épreuve, de noter immédiatement le texte des exercices et de me les envoyer (ainsi qu'à Madame Gaggioli !).

Conseils divers et variés

- Exercez-vous !
- Pratiquez du Python !!
- Ne vous jetez pas sur les corrigés, mais réfléchissez !
- Réfléchissez encore un peu !!
- Variez les plaisirs en faisant un peu d'arithmétique, un peu d'équa diff, un peu de courbes ...
- Exercez-vous à utiliser les bibliothèques de `numpy` pour manipuler les vecteurs et les matrices.
- Exercez-vous à utiliser les bibliothèques de `scipy` pour résoudre les équations différentielles.

Liens...

Le PDF à jour des différentes modifications est disponible ici : `https://goo.gl/8mK4tx`

Les différents corrigés Python ou fichiers nécessaires à certains exercices sont disponibles ici : `https://goo.gl/XiIWH7`

Pour finir, à la fin des épreuves, n'oubliez pas de m'envoyer vos exercices, impressions *etc.*
Et surtout... m** pour vos épreuves !**
Xavier Pessoles

2 Recueil d'exercices issus de la banque PT

Exercice 1 – Arithmétique

1. Soit l'entier $n = 1234$. Quel est le quotient, noté q, dans la division euclidienne de n par 10 ? Quel est le reste ? Que se passe-t-il si on recommence la division par 10 à partir de q ?
2. Écrire la suite d'instructions calculant la somme des cubes des chiffres de l'entier 1234.
3. Écrire une fonction `somcube`, d'argument `n`, renvoyant la somme des cubes des chiffres du nombre entier `n`.
4. Trouver tous les nombres entiers inférieurs à 1000 égaux à la somme des cubes de leurs chiffres.
5. En modifiant les instructions de la fonction `somcube`, écrire une fonction `somcube2` qui convertit l'entier `n` en une chaîne de caractères permettant ainsi la récupération de ses chiffres sous forme de caractères. Cette nouvelle fonction renvoie toujours la somme des cubes des chiffres de l'entier `n`.

Exercice 2 – Intégration

On cherche à calculer une valeur approchée de l'intégrale d'une fonction donnée par des points dont les coordonnées sont situées dans un fichier.

1. Le fichier `ex_01.txt`, situé dans le sous-répertoire `data` du répertoire de travail, contient une quinzaine de lignes selon le modèle suivant :

```
0.0;1.00988282142
0.1;1.07221264497
```

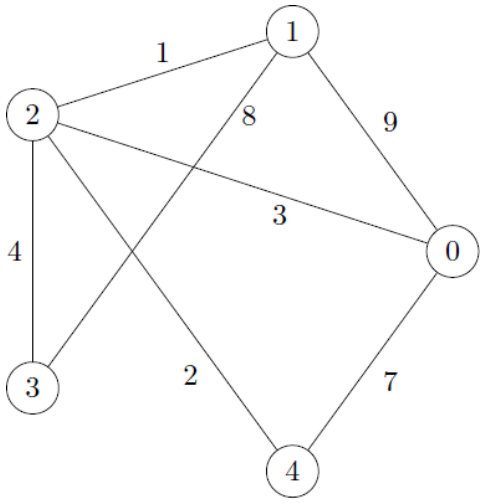
   Chaque ligne contient deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont ordonnés par abscisses croissantes. Ouvrir le fichier en lecture, le lire et construire la liste `LX` des abscisses et la liste `LY` des ordonnées contenues dans ce fichier.
2. Représenter les points sur une figure.
3. Les points précédents sont situés sur la courbe représentative d'une fonction `f`. On souhaite déterminer une valeur approchée de l'intégrale `I` de cette fonction sur le segment où elle est définie. Écrire une fonction `trapeze`, d'arguments deux listes `y` et `x` de même longueur `n`, renvoyant :

$$\sum_{i=1}^{n-1}(x_i - x_{i-1})\frac{y_i + y_{i-1}}{2}.$$

   `trapeze(LY,LX)` renvoie donc une valeur approchée de l'intégrale I par la méthode des trapèzes.
4. En utilisant la méthode d'intégration numérique `trapz` de la sous-bibliothèque `scipy.integrate` du langage Python ou la méthode `inttrap` du logiciel Scilab, retrouver la valeur approchée de l'intégrale I.

Exercice 3

On considère le graphe `G` suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière :



1. Construire la matrice $(M_{ij})_{0\leq i,j\leq 4}$, matrice de distances du graphe `G`, définie par :
   « pour tous les indices i, j, M_{ij} représente la distance entre les sommets i et j, ou encore la longueur de l'arête reliant les sommets i et j ».
   On convient que, lorsque les sommets ne sont pas reliés, cette distance vaut -1. La distance du sommet i à lui-même est, bien sûr, égale à 0.
2. Écrire une suite d'instructions permettant de dresser à partir de la matrice `M` la liste des voisins du sommet 4.
3. Écrire une fonction `voisins`, d'argument un sommet i, renvoyant la liste des voisins du sommet i.
4. Écrire une fonction `degre`, d'argument un sommet i, renvoyant le nombre des voisins du sommet i, c'est-à-dire le nombre d'arêtes issues de i.
5. Écrire une fonction `longueur`, d'argument une liste `L` de sommets de `G`, renvoyant la longueur du trajet d'écrit par cette liste `L`, c'est-à-dire la somme des longueurs des arêtes empruntées. Si le trajet n'est pas possible, la fonction renverra -1.

Exercice 4 – Gestion de liste

Soit un entier naturel n non nul et une liste `t` de longueur n dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans `t` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus de la liste `t1` suivante vaut 4 :

i	0	1	2	3	4	5	6	7
t1[i]	0	1	1	1	0	0	0	1

i	8	9	10	11	12	13	14
t1[i]	0	1	1	0	0	0	0

1. Écrire une fonction `nombreZeros(t,i)`, prenant en paramètres une liste `t`, de longueur n, et un in-

dice i compris entre 0 et $n-1$, et renvoyant :

$$\begin{cases} 0, \text{ si } t[i]=1 \\ \text{le nombre de zéros consécutifs dans t} \\ \text{à partir de t[i] inclus, si t[i]} = 0. \end{cases}$$

Par exemple, les appels `nombreZeros(t1,4)`, `nombreZeros(t1,1)` et `nombreZeros(t1,8)` renvoient respectivement les valeurs 3, 0 et 1.

2. Comment obtenir le nombre maximal de zéros contigus d'une liste `t` connaissant la liste des `nombreZeros(t,i)` pour $0 \leq i \leq n-1$? En déduire une fonction `nombreZerosMax(t)`, de paramètre `t`, renvoyant le nombre maximal de 0 contigus d'une liste `t` non vide. On utilisera la fonction `nombreZeros`.
3. Quelle est la complexité de la fonction `nombreZerosMax(t)` construite à la question précédente ?
4. Trouver un moyen simple, toujours en utilisant la fonction `nombreZeros`, d'obtenir un algorithme plus performant.

Exercice 5 – Probabilités

Soient n un entier naturel strictement positif et p un réel compris entre 0 et 1. On considère X et Y deux variables aléatoires à valeurs dans $\mathbb{N}$ sur un espace probabilisé donné. X suit une loi de Poisson de paramètre $\lambda = np$ et Y suit une loi binomiale de paramètres (n,p).

1. Définir une fonction `Px`, d'arguments k, n et p, renvoyant la valeur de $P(X=k)$. $k!$ (factorielle k) s'obtient par `factorial(k)` en Python (bibliothèque math) et `prod(1 : k)` en Scilab. Déterminer, pour $n=30$ et $p=0,1$, la liste des valeurs de $P(X=k)$ pour $k \in \mathbb{N}$, $0 \leq k \leq 30$.
2. Définir une fonction `Py`, d'arguments k, n et p, renvoyant la valeur de $P(Y=k)$. On pourra utiliser `comb` de la sous-bibliothèque `scipy.misc` en Python et `binomial` en Scilab.
   Déterminer, pour $n=30$ et $p=0,1$, la liste des valeurs de $P(Y=k)$ pour $k \in \mathbb{N}$, $0 \leq k \leq 30$.
3. Soit $k \in \mathbb{N}$. On rappelle que, sous certaines conditions sur n et p, la probabilité $P(Y=k)$ peut être approchée par $P(X=k)$. Déterminer une fonction `Ecart` d'arguments n et p, renvoyant le plus grand des nombres $|P(Y=k)-P(X=k)|$, pour $0 \leq k \leq n$.
4. Soit e un réel strictement positif. Déterminer une fonction `N`, d'arguments e et p, renvoyant le plus petit entier n tel que `Ecart(n, p)` soit inférieur ou égal à e.
5. Faire l'application numérique dans les quatre cas suivants :
   - $p=0,075$ avec $e=0,008$ et $e=0,005$;
   - $p=0,1$ avec $e=0,008$ et $e=0,005$. Interpréter le dernier résultat.

Exercice 6 – $f(x)=0$

On considère la fonction g définie sur $[0,2[$ par :

$$g(x)=\begin{cases} x & \text{pour } 0 \leq x < 1 \\ 1 & \text{pour } 1 \leq x < 2 \end{cases}$$

1. Définir la fonction g. Tracer sa courbe représentative sur $[0,2[$, c'est-à-dire la ligne brisée reliant les points $(x, g(x))$ pour x variant de 0 à 1,99 avec un pas de 0,01.
2. Définir une fonction f donnée de manière récursive sur $[0,+\infty[$ par :

$$f(x)=\begin{cases} g(x) & \text{pour } 0 \leq x < 2 \\ \sqrt{x} f(x-2) & \text{pour } x \geq 2 \end{cases}$$

3. Tracer la courbe représentative de f sur $[0,6]$.
4. Écrire les instructions permettant de calculer, à 10^{-2} près, la plus petite valeur $\alpha > 0$ telle que $f(\alpha) > 4$.

Exercice 7 – Algorithmique

On considère le code Python de la fonction `d` suivante :

Python

```
def d(n):
    L =[1]
    for nombre in range(2,n+1):
        if n%nombre == 0:
            L.append(nombre)
    return L
```

1. Quel est le résultat de l'appel `d(4)` ? Puis de l'appel `d(10)` ? Que fait la fonction `d` ?
2. Un diviseur non-trivial d'un entier `n` est un diviseur de `n` différent de 1 et de `n`. Écrire une fonction `DNT`, d'argument `n`, renvoyant la liste des diviseurs non-triviaux de l'entier `n`.
3. Écrire une fonction `sommeCarresDNT`, d'argument `n`, renvoyant la somme des carrés des diviseurs non-triviaux de l'entier `n`.
4. Écrire la suite des instructions permettant d'afficher tous les nombres entiers inférieurs à 1000 et égaux à la somme des carrés de leurs diviseurs non-triviaux. Que peut-on conjecturer ?

Exercice 8 – Chiffrer – déchiffrer

Soit n un entier vérifiant $n \leq 26$. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de n lettres. Par exemple pour $n=3$, le décalage sera le suivant :

Avant décalage	a	b	c	...	x	y	z
Après décalage	d	e	f	...	a	b	c

Le mot `oralensam` devient ainsi `rudohqvdp`.

1. Définir une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscule).
2. Écrire une fonction `decalage`, d'argument un entier `n`, renvoyant une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique, décalées de `n`, comme indiqué ci-dessus.
3. Écrire une fonction `indices`, d'arguments un caractère `x` et une chaîne de caractères `phrase`, renvoyant une liste contenant les indices de `x` dans `phrase` si `x` est une lettre de phrase et une liste vide sinon.

4. Écrire une fonction codage d'arguments un entier `n` et une chaîne de caractères `phrase`, renvoyant `phrase` codé avec un décalage de `n` lettres.
5. Comment peut-on décoder un mot codé ?

Exercice 9 – Fractale de Mandelbrot

On pose $M = 20$ et $m = 10$. À un nombre c quelconque, on associe la suite $(u_n)_{n\geq 0}$ définie par $u_0 = 0$ et $u_{n+1} = u_n^2 + c$ pour $n \geq 0$.

S'il existe, on note k le plus petit entier tel que l'on ait $0 \leq k \leq m$ et $|u_k| > M$. On définit alors la fonction f par

$$f : c \mapsto \begin{cases} k \text{ s'il existe} \\ m+1 \text{ sinon.} \end{cases}$$

1. Donner le code définissant la fonction f.
2. Tracer l'allure de la courbe représentative de la fonction f sur $[-2;2]$, en créant une liste `LX` de 401 valeurs équiréparties entre -2 et 2 inclus et en utilisant les fonctions `plot` et show de la sous-bibliothèque `matplotlib.pyplot`.
3. Construire le tableau des valeurs $f(x+iy)$ où x prend 101 valeurs comprises entre -2 et 0,5 et y prend 101 valeurs entre -1,1 et 1,1. *On rappelle que le nombre complexe i est représenté par $1j$. Par exemple, le complexe $1+2i$ est représenté par $1+2j$.*
4. Tracer l'image que code ce tableau. On pourra utiliser les fonctions `imshow` et `show` de la sous- bibliothèque `matplotlib.pyplot`. Quels paramètres peut-on modifier pour obtenir une meilleure résolution ?

Exercice 10 – Calcul matriciel

Dans cet exercice, avec Python on pourra utiliser la fonction `array` *de la bibliothèque* `numpy`*, ainsi que la fonction* `eig` *de la sous-bibliothèque* `numpy.linalg`*. Avec Scilab, on utilisera* `spec`.

1. Créer deux matrices $R = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$ et $S = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$ et les faire afficher.
2. Créer une fonction `test`, d'argument M, renvoyant la valeur `n` si M est une matrice carrée d'ordre `n` (entier naturel non nul) et zéro dans tous les autres cas. Vérifier la fonction test sur R et sur S.
3. Le fichier `ex_006.txt`, situé dans le sous-répertoire `data` du répertoire de travail, contient un tableau de valeurs flottantes. Lire ce tableau dans le fichier et vérifier qu'il correspond bien à une matrice carrée d'ordre 5 que l'on désignera par `M1`.
4. Déterminer les valeurs propres de la matrice `M1`.
5. Créer une fonction `dansIntervalle`, d'arguments une liste L et deux réels a et b, renvoyant la valeur `True` si tous les éléments de la liste L sont dans l'intervalle $[a,b]$ et `False` sinon. Vérifier que toutes les valeurs propres de la matrice `M1` sont dans l'intervalle $[0,1]$.

Exercice 11 – Tri de liste

Soit N un entier naturel non nul. On cherche à trier une liste L d'entiers naturels strictement inférieurs à N.

1. Écrire une fonction comptage, d'arguments L et N, renvoyant une liste P dont le k-ième élément désigne le nombre d'occurences de l'entier k dans la liste L.
2. Utiliser la liste P pour en déduire une fonction `tri`, d'arguments L et N, renvoyant la liste L triée dans l'ordre croissant.
3. Tester la fonction `tri` sur une liste de 20 entiers inférieurs ou égaux à 5, tirés aléatoirement.
4. Quelle est la complexité temporelle de cet algorithme ? La comparer à la complexité d'un tri par insertion ou d'un tri fusion.

Exercice 12 – Courbes paramétrées

1. Deux paramètres b et w valant respectivement 0,5 et 6,0, définir trois fonctions d'une variable t renvoyant des couples :

$$\begin{cases} p : t \mapsto & (\cos(t)+b\cos(wt), \sin(t)+b\sin(wt)) \\ v : t \mapsto & (-\sin(t)-bw\sin(wt), \\ & \quad \cos(t)+bw\cos(wt)) \\ a : t \mapsto & \big(-\cos(t)-bw^2\cos(wt), \\ & \quad -\sin(t)-bw^2\sin(wt)\big) \end{cases}$$

   Vérifier ces fonctions sur un exemple.
   $p(t) = (x(t), y(t))$ désigne la position dans le plan d'une masse ponctuelle mobile au cours du temps, $v(t) = (x'(t), y'(t))$, sa vitesse, et $a(t) = (x''(t), y''(t))$, son accélération.
2. Construire la liste `L` des points $p(t)$, pour t variant de $-\pi$ à π avec un pas de discrétisation δt vérifiant $\delta t = 0{,}01\,\pi$.
3. Faire tracer dans le plan muni d'un repère orthonormal la ligne polygonale reliant les points $p(t)$ de la liste `L`.
4. Définir puis tester la fonction c d'une variable t qui renvoie le couple des coordonnées du centre de courbure donnée par :

$$c(t) = \big(x(t) - d\,y'(t), y(t) + d\,x'(t)\big)$$

   où

$$d = \frac{x'(t)^2 + y'(t)^2}{x'(t)y''(t) - y'(t)x''(t)}.$$

5. Rajouter sur le graphique précédent la ligne décrite par les centres de courbure, avec la même discrétisation en temps.
6. Calculer la longueur de la ligne polygonale reliant les points $p(t)$, pour différents pas de discrétisation δt. Observer l'évolution de cette longueur lorsque δt diminue.

3 Exercices de la banque PT retranscrits par vos prédécesseurs – 2015

R Les exercices vous sont retranscrits tels que vos prédécesseurs nous les ont retranscrits.

Exercice 1

Nombre de manières de payer n euros avec des pièces de `L` (liste de valeurs).

Exercice 2 – Nombres riches ?

Écrire une fonction qui donne les diviseurs d'un entier n, puis qui donne les k premiers nombres riches et leur nombre de diviseurs sous forme d'une liste contenant des groupes comme `[2,2]` ? Question sur les facteurs premiers.

Exercice 3

« C'était une histoire de n listes de taille n dans une liste et je devais construire par récursivité un programme qui me donnait le nombre maximal de listes possibles de taille `p` dans ma liste de liste.»

Exercice 4

On considère la somme $S=\sum_{n=1}^{+\infty}\frac{(-1)^n}{n}$.

1. Montrer que le reste de la série vérifie $|R_n|\leq\frac{1}{1+n}$.
2. Écrire une fonction qui calcule la somme S à 10^{-6} près.
3. Tracer S_n en fonction de n (on rendra une centaine de points).

Exercice 5

Dans un ensemble $E=\{a_1,a_2,...,a_n\}$, A_i représente une partie de E et est représenté par le code c_i tel que $c_i[k]=0$ si $a_k\in A_i$ et 1 sinon.

1. Écrire la fonction `reunion` d'arguments c_1 et c_2 codes respectifs de A_1 et A_2 et retournant le code de $A_1\bigcup A_2$.
2. Écrire la fonction `estrec` d'argument `L` une liste de codes associés à des sous ensembles de E et retournant le booléen `True` su la réunion de ces sous ensembles vaut `E` ou `False` sinon.
3. Soit R un ensemble de parties de E représenté par une liste L de codes. On dit que R est un recouvrement minimal de E si la réunion des parties de R est E et que pour tout R' inclus dans R et différent de R, R' n'est pas un recouvrement de E (la réunion des parties de R' est différente de E.
4. Écrire la fonction `estRecMin` d'argument `L` une liste de codes représentant R, qui retourne le booléen `True` si R est un recouvrement minimal de E `False` sinon.

Exercice 6

Soit s définie par $s_0=1$, $s_1=1$, et pour tout n, $s_{2n}s_n+s_{n+1}$.

1. Calculer les 8 premiers termes de la suite.
2. Écrire la fonction `LS` d'argument N et renvoyant les $N+1$ premiers éléments de la suite s. Calculer $L_{128}=LS(128)$.
3. On sépare la liste L_{128} de 1 à 2, de 2 à 4, de 4 à 8... Quelle symétrie y a-t-il pour chaque paquet ? Extraire ces paquets.
4. Écrire la fonction `LP` d'argument N et renvoyant les $N+1$ premières lignes du triangle de Pascal. On rappelle que $\binom{n+1}{p}=\binom{n}{p-1}+\binom{n}{p}$ et $\binom{n}{0}=\binom{n}{n}=1$. On a par exemple, $TP(4)=[[1],[1,1],[1,2,1],[1,3,3,1],[1,4,6,4,1]$.
5. ...

Exercice 7

« Il m'était demandé de créer une liste regroupant les abscisses et ordonnées pour k allant de 1 à 12, des points complexes de module variant entre 0 et 1, d'angle $\frac{k\pi}{6}$ et ensuite de tracer ces points. Je suis arrivé à le faire mais j'ai été obligé de me faire confirmer par l'examinateur ce qu'il fallait faire car l'énoncé me bloquait. »

Exercice 8

R Exercice tombé au moins deux fois.

Un segment $[a,b]$ est noté comme une liste `[a,b]`. Soient I_1 et I_2 deux segments. Ils sont disjoints si leur intersection est vide.

1. Écrire une fonction `disjoint` d'arguments i_1 et i_2 (deux listes représentant des segments) et retournant `True` s'ils sont disjoints, `False` sinon.
2. La fusion de deux segments correspond à un segment qui a pour min le plus petit des deux min et pour max le plus grand des deux max. Écrire une fonction `fusion` d'arguments i_1 et i_2 et qui retourne la liste correspondant à la fusion de ces deux segments.
3. On dit qu'une liste de segments est bien fondée si les segments de cette liste sont disjoints et rangés dans l'ordre croissant. (On dit que $I_1=[a,b]$ est inférieur à $I_2=[c,d]$ si $b<c$.)
   (a) Les listes suivantes sont-elles bien fondées : $L=[[0,3],[6,7],[2,5]]$ et $L=[[0,1],[2,3],[4,5]]$?
   (b) Écrire une fonction récursive `verif` qui dit si une liste est bien fondée ou non.

Exercice 9

Voici un théorème d'arithmétique : tous les nombres entiers sont au plus décomposables en une somme de 9 cubes.

1. Définir une fonction `estcube(n)` qui retourne `True` si n est un cube, `False` sinon. Afficher tous les cubes inférieurs ou égaux à 250.
2. Définir une fonction `S2cube(n)` qui retourne `True` si n est la somme de deux cubes, `False` si-

non. Afficher tous les entiers inférieurs ou égaux à 250 qui sont somme de deux cubes.
3. Définir une fonction `S4cube(n)` qui retourne `True` si n est la somme de quatre cubes, `False` sinon. Afficher tous les entiers inférieurs ou égaux à 250 qui ne sont pas somme de deux cubes.
4. Définir une fonction `S8cube(n)` qui retourne `True` si n est la somme de huit cubes, `False` sinon. Afficher tous les entiers inférieurs ou égaux à 250 qui ne sont pas somme de deux cubes.
5. Expliquer pourquoi les nombres qui ne sont pas somme de 8 cubes sont sommes de 9 cubes.
6. Conclure.

Exercice 10

On dit qu'un mot est un palindrome s'il est le même lorsqu'on le dit à l'envers. Par exemple le mot «radar» est un palindrome. Le mot «nul» donne «lun» à l'nvers et n'est pas un palindrome.

1. Écrire une fonction `inverse` d'argument une chaine de caractères `mot` et renvoyant la chaîne écrite à l'envers.
2. Écrire une fonction `palindrome` d'argument une chaine de caractères `mot` et renvoyant le booléen `True` si le mot est un palindrome, `False` sinon.
3. Par extension on dit qu'un nombre est un palindrome s'il est égal au nombre obtenu en écrivant les chiffres en ordre inverse. Écrire une fonction `pat_nombre` d'argument N et renvoyant la liste des nombres palindromes inférieurs ou égaux à N.

Exercice 11

1. Créer une fonction `partage` prenant en argument une liste L et un entier a retournant 2 listes $Linf$ et $Lsup$ respectivement les listes des éléments de L inférieurs à a et et ceux supérieurs à a.
2. Créer une fonction `trirapide` utilisant la précédente.
3. Modifier les deux fonctions précédentes afin de connaître le nombre de comparaisons effectuées.
4. Tester votre fonction avec une liste de 4000 valeurs entre -999 et 999. Utiliser `randint(p,q)` pour créer cette liste.

Exercice 12

On qualifie un vecteur de creux lorsque le nombre de zéros qu'il contient est au moins supérieur à la moitié de son nombre de coordonnées. Par exemple $v = (1,2,0,4,7,0,0,0,0)$ est creux (5 zéros et 9 coordonnées). Son codage est creux est $v = [9,[1,2,4,5],[0,1,3,4]]$ (nombre de coordonnées, liste des éléments non nuls, liste des indices de ces éléments).

1. Définir la fonction `creux` d'argument une liste v représentant un vecteur et retournant un booléen indiquant si le vecteur est creux ou non.
2. Définir la fonction `coder` d'argument une liste v représentant un vecteur et retournant le codage creux du vecteur.
3. Définir une fonction `decoder`.
4. Définir une fonction `simul` d'argument le codage `C` d'un vecteur v et un scalaire a retournant le cordage creux du vecteur av.
5. Définir une fonction `coefficient(j,C)` qui renvoie la coordonnée j du vecteur creux associé au code C.

Exercice 13

1. Créer une fonction `C` d'argument un entier d et qui renvoie un entier formé par les chiffres de d dans l'ordre croissant. Par exemple, $C(1542) = 1245$.
2. Créer une fonction `D` qui fait la même chose que `C` mais dans l'ordre décroissant.
3. On définit une suite u_n par $u_0 = a \in \mathbb{N}$ et $u_{n+1} = D(u_n) - C(u_n)$. Écrire une fonction qui calcule les termes de cette suite.
4. ...

Exercice 14

1. Créer une fonction `nbits` d'argument un entier `n` qui renvoie le plus petit entier i tel que $n \leq 2^i$.
2. Créer une fonction `sansrep` d'argument une liste L d'entiers et qui retourne la liste des éléments de L en unique exemplaire. Exemple : `sansrep([5,0,0,0,3,3,5])=[5,0,5]`. On utilisera la fonction `in`.
3. Créer une fonction `position` d'argument une liste L d'entiers et e un entier, retournant la première position de e dans L (en supposant que e est dans L).
4. Retourner l'écriture binaire de la position de e dans la liste. On utilisera `nb.format(i)`.

Exercice 15

Soit n un entier fixé. On étudie le codage de listes d'entiers de longueur. On dispose d'une liste représentant le codage `C` de longueur n contenant des 0 et des 1. Une liste `E` d'entiers de longueurs n est la liste codée d'une liste A telle que A contient les éléments de la liste E dont les indices sont ceux pour lesquels `C` contient 1. Ainsi si `E=[2,4,6,7]` et `C=[0,1,1,0]` alors `A=[4,6]`.

1. Créer une fonction `decoder` d'arguments deux listes `E` et `C` qui renvoie la liste `A`.
2. Créer une fonction `coder` d'arguments `E` et `A` retournant `C`.
3. Créer une fonction `incrementer` d'argument `C` et retournant une liste `C'` contenant l'écriture binaire du nombre $N+1$ avec N nombres dont l'écriture binaire est C (si `C=[0,1,1,0]`, `N=6` donc `N+1=7` et `C'=[0,1,1,1]`.

Exercice 15

Soit T_n, le reste de la division euclidienne par 2 de la somme des chiffres représentant n en base 2. (exemple : 13 s'écrit 1101 donc on regarde le reste de la division de 1+1+0+1 par 2, donc $T(13) = 1$).

1. (a) Trouver une relation entre T_{2n} et T_n.
   (b) De-même, trouver une relation entre T_{2n+1} et T_n.

(c) Créer une fonction récursive pour calculer T_n

2. Afficher les lignes suivantes :
   - T_0, T_1
   - T_0 T_1 ; T_2 T_3 ;
   - T_0 T_1 T_2 ; T_3 T_4 T_5.

   Remarquer une particularité et en déduire une manière simple de calculer les 2^p premiers termes de la suite T_n.
3. Montrer que : $\sum_{n=0}^{+\infty} = \frac{T_n}{2^{n+1}}$ converge.
4. Calculer numériquement une valeur approchée de la somme à $10^(-6)$ près.

Exercice 16

1. Créer une fonction qui compte le nombre d'éléments pairs d'une liste.
2. Créer une fonction qui crée la liste des éléments pairs d'une liste.
3. Créer une fonction qui renvoie le booléen `True` si la liste ne contient que des chiffres pairs, `False` sinon.
4. Créer une fonction qui donne la position du maximum dans la liste.
5. Créer une fonction qui donne la médiane de la liste.
6. Créer une liste de 20 nombres aléatoires entre 0 et 100. Tester les fonctions.

Exercice 17

1. Écrire une fonction `binaire` d'argument $n \in \ltimes$, renvoyant sous forme de liste l'écriture binaire de n/
2. Écrire une fonction `nombreDeUn` d'argument $n \in \mathbb{N}$, renvoyant le nombre de 1 dans `binaire(n)`.
3. La définition du «n-palindrome» était donnée dans l'énoncé.
   (a) Écrire une fonction `Palindrome` d'argument $n \in \mathbb{N}$ permettant de déterminer si n est un 2-palindrome.
   (b) Donner les 2-palindromes entre 0 et 100.

Exercice 18

Les nombres sont écrits sous forme de chaîne de caractères.

1. Déterminer tous les entiers inférieurs à 10000 égaux à la puissance 4 de la somme de leurs chiffres.
2. Écrire une fonction `DixVersDeux` qui prend comme argument un nombre en base 10 (chaine de caractères) et envoie le nombre en base 2.
3. Écrire une fonction `DeuxVersDix` réciproque de la fonction précédente.
4. Déterminer tous les entiers inférieurs à 10 000 égaux à la puissance 4 de la somme de leurs chiffres en base 2.
5. Écrire une fonction `DixVersBase(chaine,base)`.
6. Refaire la question 4 avec les bases 3 à 9.

Exercice 18 – Bis

On écrit 18 par une chaine de caractères : `'18'` en base 10 et `'10010'` en base 2.

1. Écrire une fonction `somchi` d'argument une chaine de caractères et retournant la somme de ses chiffres (exemple `somchi('18')` retourne 9).
2. Donner la liste des nombres entiers égaux à la somme de leurs chiffres à la puissance 4.
3. Écrire la fonction `DixVersDeux`, d'argument une chaine de caractère d'un nombre en en base 10 et retournant le nombre en base 2.
4. Écrire la fonction réciproque `DeuxVersDix`.
5. Donner la liste des nombres entiers en base 2 égaux à la puissance 4 de la somme de leur chiffre.

Exercice 19

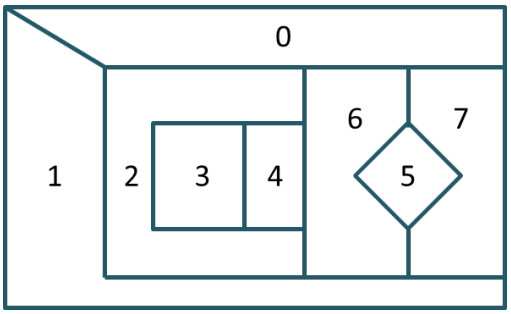
- Deux algorithmes à expliquer (l'un transforme un nombre en une liste de ses chiffres et l'autre est la fonction inverse) exemple : 2015 : alg1 $\rightarrow$ [2, 0, 1, 5] – alg2 $\rightarrow$ 2015.
- Écrire la fonction récursive `perm` d'argument une liste `L` qui renvoi toute les permutations de cette liste.
- 2 autres questions non traitées.

Exercice 20

Soit x un réel, $v_0 = x$, $a_n = \lfloor v_n \rfloor$, $v_{n+1} = \begin{cases} 0 \\ \frac{1}{a_n - 1} \end{cases}$ si $a_n > 1$.

1. Calculer les 10 premiers termes de la suite pour $x = \sqrt{3}$.
2. Définir $A(x)$, la fonction qui définit la suite et renvoie les 10 premiers termes de la suite. La tester avec $\sqrt{2}$, $\frac{1+\sqrt{5}}{2}$, $\sqrt{5}$.
3. Définir la fonction $Nb(x)$ qui renvoie le numérateur et le dénominateur de $a + \frac{b}{n}$.
4. Définir le quotient de la fonction $Q(x)$ qui renvoie le quotient $a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \ldots \cfrac{1}{a_r}}}}$ avec r le plus petit entier tel que $a_r < 1$.

Exercice 21



Une carte numérotant des pays est numérotée de 0 à i. Chaque pays se voit attribuer une couleur. Le but est que des pays voisins n'aient pas la même couleur.

1. Écrire la liste carte constituée elle-même de liste donnant pour chaque pays le numéro des pays

dont le numéro est supérieur à ce pays et ayant une frontière commune avec ce pays. Exemple : `carte[3]=[6]`, `carte[5]=[6,7]`.

2. Définir une fonction `voisins` de paramètres `(i,j,carte)` avec $i \neq j$. Cette fonction doit renvoyer un booléen `True` ou `False` si les pays i et j ont une frontière commune.
3. On attribue une couleur à chaque pays. Les couleurs sont représentées par des entiers naturels. On définit la liste `color` qui attribue pour chaque pays un numéro correspondant à sa couleur. Si le pays n'a pas de couleur, son numéro est -1.
   Définir la fonction `donnecouleur` de paramètre `p` pour le numéro de pays, `carte` et `color`. La fonction doit changer la liste couleur de façon à attribuer au pays `p` la couleur la plus petite tel qu'il ait une couleur différente de ses voisins.
   Exemple avec 3 : le pays 3 correspond à `p=2`. Admettons que les pays 2 et 4 aient respectivement les couleurs 1 et 2, on attribue à 4 la couleur 2.

Exercice 22

Soit `L` une liste triée et `a` un réel.

1. Écrire la fonction `ajout(L,a)` qui insère `a` dans `L` au bon endroit.
2. Écrire la fonction `fusion(L1,L2)` qui classe en une liste triée la somme de `L1` et `L2`.

Exercice 23

On étudie les entiers tels que $n = a^3 + b^3$.

1. Montrer que 3 ne s'écrit pas comme la somme de 2 cubes.
2. Écrire un programme afin de déterminer la liste des entiers n tels que $n = a^3 + b^3$ avec $n < 100$.
3. Écrire une fonction renvoyant un booléen nous indiquant si n est la somme de deux cubes.
4. Écrire une fonction qui indique si n est la somme de 2 cubes et l'ensemble des couples `(a,b)` tels que $n = a^3 + b^3$.

4 Exercices issus de «L'informatique pas à pas en prépa, éditions ellipses», Frédéric Butin

Exercice 1 – Opérations sur les polynômes – 2.11.3 p.50

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Un polynôme $P=\sum_{j=0}^{n} a_j X^j \in \mathbb{R}[X]$ de degré n est représenté dans cet exercice par le tableau $P=[a_0,...,a_n]$.

1. Créer une fonction `affiche_poly` qui permet d'afficher un polynôme sous la forme $P=\sum_{j=0}^{n} a_j X^j$.
2. Créer une fonction `degre_poly` qui calcule le degré d'un polynôme.
3. Implémenter la somme, le produit et la multiplication par un scalaire comme des fonctions notées `add_poly`, `mul_poly` et `mul_sca_poly`.
4. Créer une fonction `prsc_poly` qui calcule le prduit scalaire canonique de deux polynômes.
5. Créer une fonction `deriv_poly` qui calcule la dérivée d'un polynôme.

Exercice 2 – Produits polynômes – 2.11.20 p.65

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Un polynôme $P=\sum_{j=0}^{n} a_j X^j \in \mathbb{R}[X]$ de degré n est représenté dans cet exercice par le tableau $P=[a_0,...,a_n]$.

1. Créer une fonction `affiche_poly` qui permet d'afficher un polynôme sous la forme $P=\sum_{j=0}^{n} a_j X^j$.
2. Créer une fonction `degre_poly` qui calcule le degré d'un polynôme.
3. Implémenter le produit de deux polynômes. On notera `mul_poly` cette fonction. Donner sa complexité.

On suppose désormais que $n=2^k=2m$. La méthode qui suit permet de calculer le produit de deux polynômes en utilisant le principe «diviser pour régner».

On pose $P=P_1+X^m P_2$ et $Q=Q_1+X^m Q_2$, où P_1 et Q_1 sont de degré strictement inférieur à m. Ainsi, $PQ=P_1Q_1+X^m(P_1Q_2+Q_1P_2)+X^n P_2Q_2$.

1. Calculer le produit de deux polynômes de degré strictement inférieur à n revient donc à calculer 4 produits de deux polynômes de degré inférieur à $\frac{n}{2}$. Implémenter cet algorithme en une fonction `mul_poly_div`. Quelle est sa complexité ? Qu'en conclure ?
2. Une autre méthode de calcul consiste à poser $R_1=P_1Q_1$, $R_2=P_2Q_2$ et $R_3=(P_1+P_2)(Q_1+Q_2)$. Expliciter PQ en fonction des polynômes R_1, R_2, R_3. En déduire un algorithme (appelé algorithme de Karatsuba) permettant le calcul de PQ que l'on implémentera en une fonction `mul_poly_kara`. Comparer la complexité de cet algorithme à celle des algorithmes des questions précédentes.
3. Que faire quand n n'est pas de la forme 2^k.

Exercice 3 – Courbes en polaires – 4.6.25 p.111

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Pour tout $n\in\mathbb{N}^*$, on considère Γ_n en coordonnées polaires définie par :

$$\sigma_n(\theta)=\cos^3(n\theta)-\sin^3(n\theta).$$

1. Représenter la courbe Γ_0.
2. Représenter sur un même graphique les courbes Γ_j, pour $j\in[\![0,3]\!]$.

Exercice 4 – Fonction de Takagi – 4.6.26 p.112

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

La fonction de Takagi est définie sur $[0,1]$ par $T: x\mapsto \sum_{k=0}^{\infty}\frac{d\left(2^k x\right)}{2^k}$, où $d(y)$ représente la distance de y à l'entier le plus proche. On peut montrer que cette fonction est continue sur $[0,1]$ mais nulle part dérivable.

1. Pour tout entier $n\in\mathbb{N}$, majorer $\|T-T_n\|_\infty=\sup_{x\in[0,1]}|T(x)-T_n(x)|$ où $T_n: x\mapsto \sum_{k=0}^{n}\frac{d\left(2^k x\right)}{2^k}$.
2. Représenter le graphe de cette fonction, appelé la courbe du blanc-manger.

Exercice 5 – Modèle logistique – 4.6.27 p.113

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Pour tout $a\in]0,3]$, on considère la suite récurrente $(u_n)_{n\in\mathbb{N}}$ définie par $u_0\in\left[0,1+\frac{1}{a}\right]$ et pour tout $n\in\mathbb{N}^*$, $u_{n+1}=(1+a(1-u_n))u_n$. Cette suite représente, à un facteur près, la population d'une espèce.

1. Pour $a=1$ et $u_0=0,5$, représenter graphiquement les 10 premiers termes de la suite.
2. On fixe $u_0=0,5$. Créer une procédure qui reçoit en arguments a_1, c, a_2 et permet de représenter les termes u_n pour $n\in[\![100,200]\!]$ et $a=a_1+jc$, $j\in[\![0,\lfloor\frac{a_2-a_1}{c}\rfloor]\!]$ (les points sont à tracer sont des points de coordonnées (a,u_n).
3. Exécuter cette procédure avec $a_1=2$, $c=0,005$, $a_2=3$ puis avec $a_1=2.84$, $c=0,0001$, $a_2=2,86$.

Exercice 6 – Enveloppe d'une famille de droites – 4.6.28 p.115

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Doit $(D_t)_{t\in I}$ une famille de droites du plan affine, où I est un intervalle de $\mathbb{R}$. On munit le plan d'un repère, de sorte que la droite D_t a pour équation :

$$u(t)x+v(t)y+w(t)=0.$$

On suppose que les applications u, v, w sont de classe $\mathscr{C}^1$ sur I et qu'elles ne s'annulent pas en même temps.

On cherche une courbe paramétrée $f : I \to \mathbb{R}^2$ telle que pour tout $t \in I$,

- $f(t) \in D_t$;
- D_t est tangente à la courbe en $f(t)$.

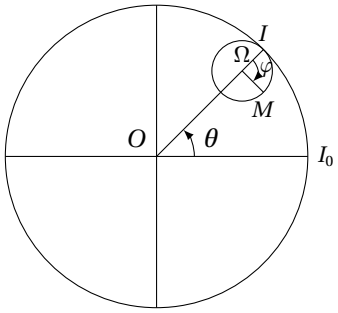
Quand elle existe, cette courbe est appelée l'enveloppe de la famille de droits $(D_t)_{t\in I}$.

1. On note $f(t) = \big(x(t), y(t)\big)$. Montrer que $\big(x(t), y(t)\big)$ est solution du système :
$$\begin{cases} u(t)x(t) + v(t)y(t) = -w(t) \\ u'(t)x(t) + v'(t)y(t) = -w'(t) \end{cases}.$$
En déduire qu'au voisinage de tout point $t_0 \in I$ tel que :
$$\begin{vmatrix} u(t_0) & v(t_0) \\ u'(t_0) & v'(t_0) \end{vmatrix} \neq 0$$
, le système précédent a une unique solution, donnée par :
$$x(t) = \frac{\begin{vmatrix} -w(t) & v(t) \\ -w'(t) & v'(t) \end{vmatrix}}{\begin{vmatrix} u(t) & v(t) \\ u'(t) & v'(t) \end{vmatrix}}, y(t) = \frac{\begin{vmatrix} u(t) & -w(t) \\ u'(t) & -w'(t) \end{vmatrix}}{\begin{vmatrix} u(t) & v(t) \\ u'(t) & v'(t) \end{vmatrix}}.$$
2. Déterminer une paramétrisation de l'enveloppe E de la famille des droites $(D_t)_{t\in\mathbb{R}}$ d'équation :
$$\sin(t)x - \cos(t)y - \sin^2(t) = 0.$$
3. Représenter, sur un même graphique, E et plusieurs droites D_t.

Exercice 7 – Hypocycloïde – 4.6.29 p.117

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Un cercle $\Gamma(\Omega, r)$ roule sans glisser à l'intérieur du cercle $C(O, R)$ (où $R > r$). On note $M = M(\theta)$ un point de Γ dont étudie la trajectoire. On note θ l'angle $\left(\widehat{\overrightarrow{i}, \overrightarrow{O\Omega}}\right)$ et φ l'angle $\left(\widehat{\overrightarrow{\Omega I}, \overrightarrow{\Omega M}}\right)$. Initialement, Ω est situé sur l'axe horizontal et M est situé en I_0.



1. Montrer que l'affixe de M est donnée par $z(\theta) = (R-r)\exp(i\theta) + r\exp(im\theta)$ où $m = 1 - \dfrac{R}{r}$. Ainsi, M a pour coordonnées :
$$\begin{cases} x(\theta) = (R-r)\cos\theta + r\cos(m\theta) \\ y(\theta) = (R-r)\sin\theta + r\sin(m\theta) \end{cases}.$$
2. On choisit $R = 4$ et $r = \dfrac{R}{4}$. Représenter la trajectoire de M. La courbe obtenue est appelée astroïde.
3. On choisit $R = 4$ et $r = \dfrac{R}{p}$ où $p \in \mathbb{N}$. Représenter, pour différentes valeurs de p, $\Gamma(\Omega, r)$ roulant sur $C(O, R)$, ainsi que la trajectoire de M. La courbe obtenue est appelée hypocycloïde à p rebroussements.
4. Vérifier que ces points sont effectivement des points de rebroussement.

Exercice 8 – Ensembles de Mandelbrot et de Julia – 4.6.30 p.119

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

L'ensemble de Mandelbrot est la partie M du plan complexe définie par $M = \{c \in \mathbb{C} /$ la suite $(z_n)_{n\in\mathbb{N}}$ définie par $z_0 = 0$ et $z_{n+1} = z_n^2 + c$ est bornée $\}$.

De même, pour tout $c \in \mathbb{C}$, l'ensemble de Julia de paramètre c est défini par $J_c = \{z \in \mathbb{C} /$ la suite $(z_n)_{n\in\mathbb{N}}$ définie par $z_0 = z$ et $z_{n+1} = z_n^2 + c$ est bornée $\}$.

On souhaite représenter l'ensemble de Mandelbrot. On fixe un entier p assez grand, et pour chaque point $c \in \mathbb{C}$, on s'intéresse à la suite $(z_n)_{n\in\mathbb{N}}$ définie par $z_0 = 0$ et pour tout $n \in \mathbb{N}$, $z_{n+1} = z_n^2 + c$. On considère que cette suite n'est pas bornée s'il existe $k \leq p$ tel que $|z_k| \geq 4$.

1. Représenter l'ensemble de Mandelbrot. On pourra utiliser la fonction `imshow` qui permet de représenter, par une couleur différente, chaque valeur de k_0, où k_0 est le plus petit entier tel que $|z_{k0}| \geq 4$.
2. En procédant de même, représenter l'ensemble de Julia J_c pour différentes valeur de c.

Exercice 9 – Courbe de Peano – 4.6.31 p.122

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

La courbe de Peano est construite à partir d'un motif de base dans le lequel on remplace chacun des 9 segments par le motif complet auquel on a appliqué une homothétie de rapport $\dfrac{1}{3}$.

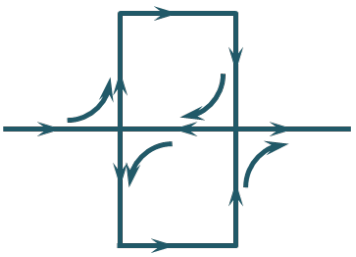
1. S'approprier le module `turtle` en réalisant un cercle avec la tortue.
2. En utilisant la tortue de Python, écrire une procédure récursive qui reçoit un entier n et trace la courbe obtenue en itérant n fois le procédé décrit ci-dessus.

R Pour utiliser le module `turtle` :

- importer le module : `import turtle` ;
- cacher la tortue : `turtle.hideturtle()` ;
- choisir la vitesse de la tortue : `turtle.speed(10)` ;
- faire en sorte que la tortue laisse un trait sur son chemin : `tortue = turtle.Pen()` ;

Oral

- faire avancer la tortue de 5 : `tortue.forward(5)` ;
- faire tourner la tortue de 90 degrés vers la gauche : `tortue.left(90)`.



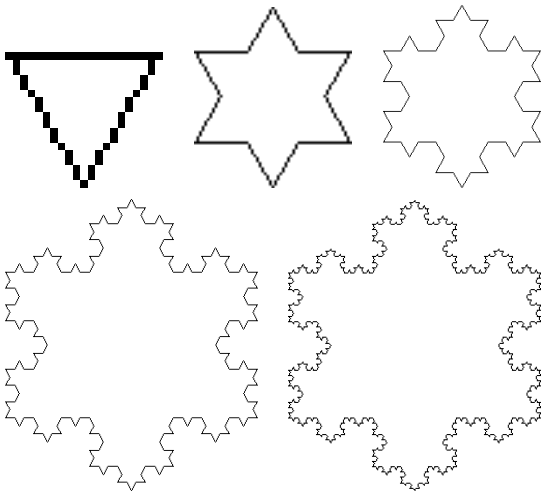
Exercice 10 – Flocon de Koch – 4.6.32 p.123

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Le flocon de Koch est construit à partir d'un triangle équilatéral sur chacun des trois côtés duquel on applique les transformations suivantes :

- on divise le côté en trois segments de même longueur ;
- on construit un triangle équilatéral ayant pour base le segment du milieu dont on supprime la base.

1. Si cela n'a pas été fait, s'approprier le module `turtle` en réalisant un cercle avec la tortue.
2. En utilisant la tortue de Python, écrire une procédure récursive qui reçoit un entier n et trace le flocon de Koch en itérant n fois le procédé décrit ci-dessus.



Exercice 11 – Intégration numérique – 4.6.33 p.124

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

On considère la fonction $f : x \mapsto \ln x$.

1. Calculer $I = \int_1^4 f(x)\mathrm{d}x$.
2. Comparer l'erreur entre I et la valeur approchée de I obtenue par les méthodes des rectangles et des trapèzes en approchant un nombre de points n où n parcourt la suite $[10, 20, 40, 100, 200, 400, 500, 600, 700, 800, 900, 1000, 5000, 10000, 20000, 100000]$.
3. Représenter graphiquement cette erreur en prenant une échelle log – log. Expliquer les graphes obtenus.

R Tracé en diagramme log – log : `plt.loglog(x,y)`.

Exercice 12 – Équation différentielle – 4.6.34 p.125

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Pour $n \in \mathbb{N}$, on considère l'équation différentielle :

$$x''(t) + 10x'(t) - x(t) = \sin(nt).$$

1. Résoudre cette équation différentielle avec les conditions initiales $x(0) = 0$ et $x'(0) = 1$.
2. Représenter le graphe des solutions pour $n \in [\![0, 10]\!]$ et $t \in [0, 7]$.

Exercice 13 – Suite de Fibonacci – 4.6.41 p.135

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

La suite de Fibonacci est la suite $(F_n)_{n\in\mathbb{N}}$ définie par $F_0 = 1$, $F_1 = 1$ et pour tout $n \in \mathbb{N}$, $F_{n+2} = F_{n+1} + F_n$.

1. Écrire une fonction qui calcule F_n à l'aide de produits de matrice.
2. Déterminer la complexité de la procédure de la question précédente en nombre d'additions et en nombre de multiplications. On distinguera les cas où :
   - la méthode d'exponentiation naïve est utilisée ;
   - la méthode d'exponentiation rapide est utilisée.

Exercice 14 – Produits de matrices – 4.6.42 p.136

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

1. Calculer la complexité, en termes d'opérations sur les coefficients, de l'addition et de la multiplication de deux matrices de $M_n\mathbb{R}$ par la méthode naïve qui consiste à utiliser la définition du produit.
   On suppose désormais que $n = 2^k = 2m$.
2. Une méthode de calcul de produit de deux matrices qui repose sur le principe « diviser pour régner » est la suivante. On pose :

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \quad \text{et} \quad N = \begin{bmatrix} N_{11} & N_{12} \\ N_{21} & N_{22} \end{bmatrix}.$$

On a donc :

$$P = MN = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}$$

avec :

$$P_{11}=M_{11}N_{11}+M_{12}N_{21} \quad P_{12}=M_{11}N_{12}+M_{12}N_{22}$$
$$P_{21}=M_{21}N_{11}+M_{22}N_{21} \quad P_{22}=M_{21}N_{12}+M_{12}N_{22}.$$

Ainsi calculer le produit de deux matrices $M_n\mathbb{R}$ revient à calculer 8 produits de deux matrices de $M_{n/2}\mathbb{R}$.
Implémenter cet algorithme en une fonction `prod_div`. Quelle est sa complexité ? Qu'en conclure ?

3. Une autre méthode de calcul consiste à poser $C_1 = M_{11}(N_{12}-N_{22})$, $C_2 = N_{22}(M_{11}+M_{12})$, $C_3 = N_{11}(M_{21}+M_{22})$, $C_4 = M_{22}(N_{21}-N_{11})$, $C_5 = (M_{11}+M_{22})(N_{11}+N_{22})$, $C_6=(M_{12}-M_{22})(N_{21}+N_{22})$ et $C_7 = (M_{11}-M_{21})(N_{11}+N_{12})$. Expliciter P_{11}, P_{12}, P_{21} et P_{22} en fonction des matrices C_1, ..., C_7. En déduire un algorithme (appelé algorithme de Strassen) permettant le calcul de $P = MN$ que l'on implémentera en une fonction `prod_strassen`. Comparer la complexité de cet algorithme à celle des algorithmes des questions précédentes.
4. Que faire quand n n'est pas de la forme 2^k ?

Exercice 15 – Équations différentielles – 4.6.46 p.144

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

On souhaite résoudre par différences finies l'équation :

$$u''(x)+2u(x)=2\cos x+\sin^2 x$$

sur $[0,\pi]$ avec les conditions aux limites $u(0)=0$ et $u(\pi)=0$.

On considère la subdivision $0=x_0<x_1<\ldots<x_n<x_{n+1}=\pi$ de l'intervalle $[0,\pi]$ où $x_i=ih$ et $h=\dfrac{\pi}{n+1}$. Pour tout i, on note u_i la valeur approchée cherchée de $u(x_i)$ et on approche la dérivée seconde $u''(x)$ par :

$$\frac{u_{i-1}-2u_i+u_{i+1}}{h^2}.$$

1. Montrer que le système d'équation obtenu s'écrit matriciellement $AU=F$, où

$$A=\begin{bmatrix} 2h^2-2 & 1 & & & \\ 1 & \ddots & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 2h^2-2 \end{bmatrix}$$

$$F=\begin{bmatrix} h^2f(x_1) \\ \vdots \\ h^2f(x_n) \end{bmatrix} \quad U=\begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix}$$

avec $f : x \mapsto 2\cos x+\sin^2 x$.
2. Résoudre le système linéaire et représenter graphiquement la solution.
3. La solution exacte de l'équation est :

$$u(x) = \frac{\sin\left(\sqrt{2}x\right)\left(3+5\cos\left(\sqrt{2}\pi\right)\right)}{2\sin\left(\sqrt{2}\pi\right)} - \frac{5\cos\left(\sqrt{2}x\right)}{2}+\frac{\cos x}{2}(4+\cos x).$$

Superposer le graphe de la solution exacte au graphe de la solution approchée.

Exercice 16 – Équations des ondes – 4.6.47 p.146

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

On souhaite résoudre par différences finies l'équation des ondes :

$$\partial_{tt}u(x,t)-c^2\partial_{xx}u(x,t)=0$$

sur $[-5,5]\times[0,0.1]$ avec les conditions initiales $u(x,0)=\exp\left(-10x^2\right)$ et $\partial_t u(x,0)=0$, et les conditions aux limites $u(-5,t)=u(5,t)=0$.

On considère la subdivision $-5=x_0<x_1<\ldots<x_n<x_{n+1}=5$ (resp. $0=t_0<t_i<\ldots<t_{m-1}<t_m=0,1$) de l'intervalle $[-5,5]$ (resp. $[0,0.1]$) où $x_i=-5+ih$ (resp. $t_k=kl$) et $h=\dfrac{10}{n+1}$ (resp. $l=\dfrac{0,1}{m}$).

Pour tout $(i,k)\in[\![0,n+1]\!]\times[\![0,m]\!]$, on note $z_{i,k}$ la valeur approchée cherchée de $u(x_i,t_k)$ et l'on approche la dérivée première $\partial_t u(x_i,t_k)$ par $\dfrac{z_{i,k+1}-z_{i,k}}{l}$ et la dérivée seconde $\partial_{tt}u(x_i,t_k)$ par $\dfrac{z_{i,k+1}-2z_{i,k}+z_{i,k-1}}{l^2}$. On choisit $n=m=100$ et $c=31,6$.

1. Résoudre l'équation par différences finies en utilisant le schéma explicite, qui consiste à approcher la dérivée seconde $\partial_{xx}u(x_i,t_k)$ par l'expression $\dfrac{z_{i-1,k}-2z_{i,k}+z_{i+1,k}}{h^2}$.
2. Créer une animation permettant de visualiser la solution obtenue quand t varie.
3. Résoudre l'équation par différences finies en utilisant le schéma implicite, qui consiste à approcher la dérivée seconde $\partial_{xx}u(x_i,t_k)$ par l'expression : $\dfrac{z_{i-1,k+1}-2z_{i,k+1}+z_{i+1,k+1}}{h^2}$.
4. Créer une animation permettant de visualiser la solution obtenue quand t varie.

Exercice 17 – Position d'une membrane – 4.6.49 p.153

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

On considère une membrane élastique dont le bord est fixé à un cadre carré horizontal. La membrane est soumise à une force verticale f. On peut montrer que l'altitude$z(x,y)$ de la membrane en un point du carré de coordonnées (x,y) vérifie l'équation de Laplace :

$$\Delta z(x,y)=f(x,y)$$

avec les conditions aux limites $z(x,y)=0$ sur le bord du carré.

On se place dans le cas où le cadre est le carré $[0,1]^2$ et où f est constante, par exemple $f(x,y)=-4$ pour tout $(x,y)\in[0,1]^2$.

Pour résoudre cette équation par la discrétisation, on pose $h=\dfrac{1}{n+1}$ et pour tout $(i,j)\in[\![0,n+4]\!]^2$, $x_i=ih$ et $y_j=jh$: on obtient ainsi un quadrillage du carré $[0,1]^2$. On a déjà $z(x_i,y_j)=0$ si $i\in\{0,n+1\}$ ou $j\in\{0,n+1\}$ par hypothèse. On note $z_{i,j}$ l'approximation cherchée de $z(x_i,y_j)$ par $z_{i,j}$ et le laplacien $\Delta z(x_i,y_j)$ par son approximation à 5 points :

$$\frac{1}{h^2}\left(z_{i-1,j}+z_{i+1,j}+z_{i,j-1}+z_{i,j+1}-4z_{i,j}\right).$$

On résout alors le système formé par les équations obtenues, qui est un système linéaire dont les inconnues sont les $z_{i,j}$ pour $(i,j)\in[\![0,1]\!]^2$.

1. Étant donné $A=(a_{i,j})_{(i,j)\in[\![1,n]\!]^2}$ et $B=(b_{i,j})_{(i,j)\in[\![1,n]\!]^2}$ deux matrices de $M_n\mathbb{R}$, le produit de Kronecker de A par B est la matrice de $M_{n^2}\mathbb{R}$ définie par :

$$A\otimes B=\begin{bmatrix} a_{1,1}B & a_{1,2}B & \dots & a_{1,n}B\\ a_{2,1}B & a_{2,2}B & \dots & a_{2,n}B\\ \vdots & & & \vdots\\ a_{n,1}B & a_{n,2}B & \dots & a_{n,n}B\end{bmatrix}.$$

   La commande `kron` de numpy permet de calculer le produit de Kronecker. Observer que le système linéaire peut se mettre sous la forme $(A_n\otimes I_n+I_n\otimes A_n)Z=F$ où A_n est la matrice du laplacien de taille n, c'est à dire :

$$A_n=\begin{bmatrix} 2 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1\\ & & -1 & 2\end{bmatrix}.$$

2. Résoudre l'équation par discrétisation, comme expliqué ci-dessus.
3. Représenter la membrane en 3d.

Exercice 18 – Polygones orthogonaux – 4.6.50 p.155

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

1. Pour $n=16$, créer en Python la base canonique de $\mathbb{R}_n[X]$.
2. Définir le produit scalaire $\varphi : (P,Q)\mapsto \int_{-1}^{1} P(t)Q(t)\mathrm{d}t$ et la norme euclidienne associée sur $\mathbb{R}_n[X]$.
3. En utilisant le procédé de Gram-Schmidt, orthonormaliser la base canonique pour le produit scalaire φ.
4. Calculer la projection orthogonale de $h : x\mapsto \cos(4\pi x)$ sur $\mathbb{R}_n[X]$ (approximation de h au sens des moindres carrés).

Exercice 19 – Système proies – prédateurs – 4.6.37 p.129

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Le système proies-prédateurs est régit par les équations de Volterra qui forment le système non linéaire :

$$\begin{cases} x'(t)=a\,x(t)-b\,x(t)y(t)\\ y'(t)=-c\,y(t)+d\,x(t)y(t)\end{cases},$$

où a, b, c et d sont des réels strictement positifs.

1. On pose $X=(x,y)^t$. Créer une fonction f qui à (X,t,a,b,c,d) fait correspondre $(a\,x-b\,x\,y,-c\,y+d\,x\,y)$.
2. Résoudre le système dans les conditions suivantes : $X=(1,2)$ et $a,b,c,d=1,1,1,1$.
3. Représenter x et y sur un même graphique.
4. Représenter les courbes paramétrées par x et y, qui donne l'évolution des deux populations.

Exercice 20 – Transformée de Fourier – 4.6.65 p.241.

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Implémenter en Python l'algorithme naïf de calcul de la transformée de Fourier d'un vecteur de $\mathbb{C}^N$.

Exercice 21 – 6.5.66 p.242.

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Soit $f:\mathbb{R}\to\mathbb{R}$ l'application 2π-périodique définie par :

$$f(x)=\begin{cases} 1 & \text{si} \quad x\in[0,\pi[\\ 10 & \text{si} \quad x\in[\pi,2\pi[\end{cases}.$$

1. En utilisant la FFT, calculer des valeurs approchées des coefficients de Fourier de f.
2. Représenter sur un même graphique f et la somme approchée de sa série de Fourier.

Exercice 22 – Débruitage d'un signal – 6.5.65 p.243.

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

On considère un signal donné par l'application :

$$f:t\mapsto 3\cos(3t)+2\sin(2t)-\frac{1}{2}\cos t$$

échantillonné sur $n=100$ points équidistants $(t_k)_{k\in[\![1,n]\!]}$ du segment $[0,2\pi]$. On introduit sur ce signal un bruit donné par une loi uniforme sur $[-1,1]$: ainsi pour tout point t_k d'échantillonnage, la valeur du signal n'est pas $f(t_k)$, mais $g(t_k)=f(t_k)+b_k$, où b_k est une réalisation d'une loi uniforme sur $[-1,1]$.

1. Représenter sur un même graphique le signal f et le signal bruité g.

2. Appliquer la FFT au signal bruité. Construire alors un nouveau signal débruité h en supprimant les fréquences d'indices trop grands (on peut par exemple ne garder qu'un quart des fréquences).
3. Représenter h sur le même graphique que f et g.

Exercice 23 – Taches solaires – 4.6.65 p.244.

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

Nous nous intéressons ici aux mesures de l'activité des taches solaires de l'année 1700 à l'année 2004. On appelle nombre de Wolf le nombre de taches solaires observées en une année.

La transformée de Fourier, qui permet de passer du domaine temporel au domaine fréquentiel, donne la possibilité de savoir s'il existe une fréquence prédominante, c'est-à-dire si les données sont périodiques.

1. Représenter graphiquement le nombre Wolf en fonction de l'année.
2. Effectuer la FFT de la liste des nombres de Wolf. On la note Y.Représenter alors $\Im(Y)$ en fonction de $\Re(Y)$.
3. On souhaite construire un chronogramme, c'est-à-dire une graphe de la puissance du signal en fonction de la fréquence (la puissance de la FFT étant égale au carré du signal de la FFT).
Tracer ce chronogramme en utilisant uniquement les coefficients de Y compris entre 1 et $E\left(\frac{n}{2}\right)$, où n est le nombre de relevés (les coefficients situés au-delà de $E\left(\frac{n}{2}\right)$ correspondant à des coefficients de Fourier d'indices négatifs) : la plage de fréquences est donc :

$$\left(\frac{k}{n}\right)_{k\in[\![1,E\left(\frac{n}{2}\right)]\!]} = \left[\frac{1}{n}, \frac{2}{n}, \dots E\left(\frac{n}{2}\right)\frac{1}{n}\right].$$

Tracer le graphe de la puissance du signal en fonction de la période. Que constate-t-on ?

Exercice 24 – Traitement d'image – Filtre – 6.5.69 p.247.

D'après Frédéric Butin, l'informatique pas à pas en prépa, éditions ellipses.

1. Créer une fonction qui permet d'afficher la composante rouge d'une image.
2. Créer une fonction qui permet d'afficher la composante cyan.

5 Corrigés – Banque PT

Exercice 1 – Arithmétique – Corrigé

```
# Question 1
n = 1234
q = n//10
r =  n%q

# r contient le nombre d'unités de n
```

```
# Question 2
s=0
while n!=0:
    q=n//10
    r = n%10
    #print(r)
    s=s+r**3
    n=q
```

```
# Question 3
def somcube(n):
    """
    Entrées :
     * n, int : nombre
    Sortie :
     * s, int : somme des cubes du chiffre n
    """
    s=0
    while n!=0:
        q=n//10
        r = n%10
        s=s+r**3
        n=q
    return s
```

```
# Question 4
res = []
for i in range (10001):
    if i == somcube(i):
        res.append(i)
```

```
# Question 5
def somcube2(n):
    """
    Entrées :
     * n, int : nombre
    Sortie :
     * s, int : somme des cubes du chiffre n
    """
    nombre=str(n)
    s=0
    for chiffre in nombre :
        s = s+int(chiffre)**3
    return s

print(somcube2(1234))
```

Exercice 2 – Intégration – Corrigé

```
# Question 1
# ==========
# Le répertoire courant est Exercice_02.
# Le sous-répertoire data contient le
# fichier ex_01.txt.

# On ouvre le fichier en lecture)
fid = open("data\ex_01.txt")

# On charge le fichier dans une liste.
# Chaque élément de la liste correspond à
# chaque ligne sous forme de chaîne de caractère.
file = fid.readlines()
# On ferme le fichier
fid.close()

LX=[]
LY=[]
for ligne in file :
    ligne = ligne.split(';')
    LX.append(float(ligne[0]))
    LY.append(float(ligne[1]))
```

```
# Question 2
# ==========
# Ne pas oublier de charger préalablement
# import matplotlib.pyplot as plt

plt.plot(LX,LY)
plt.show()
```

```
# Question 3
# ==========
def trapeze(x,y):
    res = 0
    for i in range(1,len(LX)):
        res = res+(LX[i]-LX[i-1])*0.5*(LY[i]+LY[i-1])
    return res
print(trapeze(LX,LY))
```

```
# Question 4
# ==========
from scipy.integrate import trapz
# Attention àl'ordre des arguments dans
# la fonction trapz : les_y puis les_x
# Après l'import, help(trapz) permet d'avoir
# de l'aide sur la fonction.
print(trapz(LY,LX))
```

Exercice 3 – Graphe – Corrigé

```
# Question 1
# ==========
# Matrices avec des listes
M=[[0,9,3,-1,7],
      [9,0,1,8,-1],
      [3,1,0,4,2],
      [-1,8,4,0,-1],
      [7,-1,2,-1,0]]
```

```
# Question 2 & 3
# ==============
def voisins(M,i):
    """
    Entrées :
      * M(lst) : graphe
      * i : noeud considéré
    Sortie :
      * v(lst) : liste des voisins
    """
    v = []
    # On cherche les voisins sur une ligne
    # (on pourrait le faire sur une colonne)
    for j in range(len(M[i])):
        if M[i][j]>0:
            v.append(j)
    return v

# print(voisins(M,0))
```

```
# Question 4
# ==========
def degre(M,i):
    """
    Entrées :
      * M(lst) : graphe
      * i : noeud considéré
    Sortie :
      * (int) : nomnbre de voisins
    """
    return len(voisins(M,i))
```

```
# Question 5
# ==========
def longueur(M,chemin):
    l = 0
    for i in range(len(chemin)-1):
        if M[chemin[i]][chemin[i+1]]<0:
            return -1
        else :
            l=l+M[chemin[i]][chemin[i+1]]
    return l

chemin = [1,2,3,1,4]
print(longueur(M,chemin))
chemin = [0,4,2,1,0]
print(longueur(M,chemin))
```

Exercice 4 – Corrigé

```
# Question 1
# ==========
def nombreZeros(t,i):
    if t[i]==1:
        return 0
    else :
        res = 1
        j=i+1
        while j<len(t) and t[j]==0:
            res = res+1
            j=j+1
    return res
# t1=[0,1,1,1,0,0,0,1,0,1,1,0,0,0,0]
```

```
# print(nombreZeros(t1,4))
# print(nombreZeros(t1,1))
# print(nombreZeros(t1,8))
```

```
# Question 2
# ==========
def nombreZerosMax(t):
    max=nombreZeros(t,0)
    for i in range(1,len(t)):
        tmp = nombreZeros(t,i)
        if tmp>max:
            max = tmp
    return max
print(nombreZerosMax(t1))
```

La complexité est quadratique ($\mathcal{O}^2$) du fait de la boucle `for` et de la boucle `while` imbriquée.

Pour diminuer la complexité, il est possible de parcourir une seule fois la liste. On lit alors les termes un à un. Quand on détecte un zéro, on compte alors le nombre de zéros consécutifs et on poursuit jusqu'à la fin...

Exercice 5 – Corrigé

D'après Mme Barré `http://www.lycee-lesage.net/`.

```
import math
import scipy.misc as scim
# Question 1
# ==========
def Px_poisson(k, n, p):
    """
    Entrée :
     * k(int)
     * n(int) : strictement positif
     * p(flt) : réel compris entre ]0, 1[
    X suit une loi de Poisson P(n*p)
    Sortie :
     * flt : (np)^k exp(-np)/k!
    """
    np = n * p
    return (np)**k * math.exp(- np) / math.factorial(k)

n, p = 30, 0.1
lst_px = [Px_poisson(k, n, p) for k in range(n + 1)]
```

```
# Question 2
# ==========
def Py_binomiale(k, n, p):
    """
    Entrée :
     * k(int)
     * n(int) : strictement positif
     * p(flt) : réel compris entre ]0, 1[
    Y suit une loi binomiale B(n,k)
    Sortie :
     * int : B(n,k)*p^k *(1-p)^(n-k)
    """
    return scim.comb(n,k)*p**k*(1-p)**(n-k)

n, p = 30, 0.1
lst_py = [Py_binomiale(k, n, p) for k in range(n + 1)]
```

```
# Question 3
# ==========
def ecarts(n,p):
       """
       Entrée :
    * n(int) : strictement positif
    * p(flt) : réel compris entre ]0, 1[
   Sortie :
    * maxi(flt) écart maxi entre P(Y=k) et P(X=k)
   """

       lst_px = [Px_poisson(k, n, p) for k in range(n + 1)]
   lst_py = [Py_binomiale(k, n, p) for k in range(n + 1)]
   maxi = max([abs(lst_py[i]-lst_px[i]) for i in range(len(lst_px))])
   return maxi

#print(ecarts(n,p))
```

```
# Question 4
# ==========
def E(e,p):
       """
       Retourne le plus petit entier naturel n tq ecart(n,p)<=e
       Entrée :
        * e(flt) : >0
        * p(flt) : réel compris entre ]0, 1[
   Sortie :
    n(int)
       """
       n=1
       while (ecarts(n,p))>e :
              n=n+1
       return n
```

```
# Question 5
# ==========
print(E(0.008,0.075))
print(E(0.005,0.075))
print(E(0.008,0.1))
#print(E(0.005,0.1))
```

Exercice 6 – Corrigé

```
# Import de fonctions
import matplotlib.pyplot as plt
from math import sqrt
# Question 1
# ==========
def g(x):
    if x>= 0 and x<1 :
        return x
    elif x>1 and x<2 :
        return 1
xx = [0]
t=0
while t<=1.99:
    t=t+0.01
    xx.append(t)
yy = [g(x) for x in xx]
plt.plot(xx,yy)
plt.show()
```

```
# Question 2
# ==========
def f(x):
    if x>=0 and x<2 :
        return g(x)
    else : # x>=2
        return sqrt(x)*f(x-2)
```

```
# Question 3
# ==========
xxx = [0]
t=0
while t<=6:
    t=t+0.01
    xxx.append(t)
yyy = [f(x) for x in xxx]
plt.plot(xxx,yyy)
plt.show()
```

```
# Question 4
# ==========
# On cherche àrésoudre f(x)-4 = 0 sur
# l'intervalle [5,6]
def h(x):
    res = f(x)-4
    return res

a = 5.
b = 6.
while (b-a)>0.01:
    m=(a+b)/2
    if h(m)>0:
        b=m
    else :
        a=m
m=(a+b)/2

if h(m)<0:
    m= m+ abs(b-a)
print(m,h(m))
```

Exercice 7 – Corrigé

```
# Question 1
# ==========
def d(n):
    """
    Retourne la liste de tous les diviseurs de n.
    Entrée :
     * n(int) : entier.
    Sortie :
     * L(lst) : liste des diviseurs de n.
    """
    L =[1]
    for nombre in range(2,n+1):
        if n%nombre == 0:
            L.append(nombre)
    return L
print(d(4),d(10))
```

```
# Question 2
# ==========
def DNT_01(n):
    return d(n)[1:-1]
def DNT_02(n):
    L =[]
    for nombre in range(2,n):
        if n%nombre == 0:
            L.append(nombre)
    return L
print(DNT_01(4),DNT_02(4))
print(DNT_01(10),DNT_02(10))
```

```
# Question 3
# ==========
def sommeCarresDNT_01(n):
    L = DNT_01(n)
    res = [x**2 for x in L]
    return sum(res)
def sommeCarresDNT_02(n):
    L = DNT_01(n)
    res = 0
    for x in L:
        res = res + x*x
    return res
def sommeCarresDNT_03(n):
    L = DNT_01(n)
    res = 0
    for i in range(len(L)):
        res = res + L[i]**2
    return res
print(sommeCarresDNT_01(15),sommeCarresDNT_02(15),
    sommeCarresDNT_03(15))
```

```
# Question 4
# ==========
from math import sqrt
for i in range(1001):
    if i == sommeCarresDNT_01(i) :
        print(str(i)+"\t"+str(sqrt(i)))
# Conjecture les nombres recherchés sont
# les carrés des nombres premiers.
```

Exercice 8 – Corrigé

```
# Question 1
# ==========
chaine = "abcdefghijklmnopqrstuvwxyz"
```

```
# Question 2
# ==========
def decalage(chaine,n):
    chaine = chaine[n:-1]+chaine[0:n]
    return chaine
print(chaine,decalage(chaine,3))
```

```
# Question 3
# ==========
def indices(x,phrase):
    """
    Recherche des indices de x dans phrase
    Entrée :
     * x(str) : un caractère
     * phrase(str)
    Sortie :
     * res(lst) : liste des indices de x
    """
    res = []
    for i in range(len(phrase)):
        if phrase[i] == x:
            res.append(i)
    return res

print(indices("a","akjlkjalkjlkjalkjlkja"))
```

```
# Question 4
# ==========
def codage(n,phrase):
    ch = "abcdefghijklmnopqrstuvwxyz"
    ch_c = decalage(ch,n)
    print(ch_c)
    phrase_c=""
    for c in phrase :
        i = indices(c,ch)
        i = i[0]
        phrase_c = phrase_c+ch_c[i]
    return phrase_c
print(codage(3,"oralensam"))
```

```
# Question 5
# ==========
# Solution 1 : essayer les 26 permutations,
# jusqu'à trouver une phrase qui est du sens.
# Solution 2 : statistiquement le e est la lettre
# la plus présente dans la langue française. On
# peut donc déterminer la fréquence d'apparition
# des lettres. # La lettre la plus fréquente
# peut être assimilée au "e".
# On calcule ainsi le décalage...
```

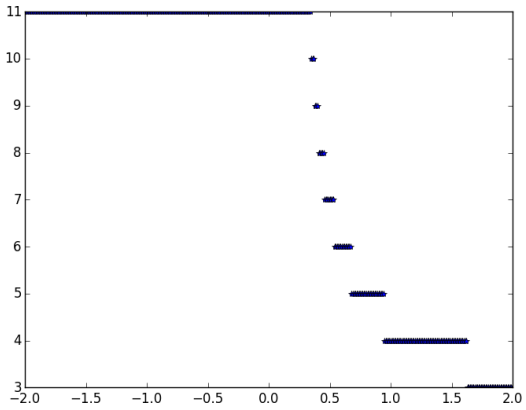
Exercice 9 – Fractale de Mandelbrot – Corrigé

```
# Question 1
# ==========
def suite_u(c,n):
    """
    Calcul de la suite u au rang n.
    Entrées :
     * c(flt) : nombre quelconque
     * n(int)
    Sortie :
     * res(flt) : valeur de u(n)
    """
    res = 0
    i=0
    while i!=n:
        res = res*res+c
        i=i+1
    return res
```

```
def recherche_k(m,M,c):
    """ Recherche de k """
    k=0
    while k<=m:
        if abs(suite_u(c,k))>M:
            return k
        k=k+1
    return -1
```

```
def fonction_f(m,M,c):
    """ Fonction """
    k = recherche_k(m,M,c)
    if k>=0:
        return k
    else :
        return m+1
```

```
# Question 2
#===========
import matplotlib.pyplot as plt
m,M=10,20
LX = [-2+4*x/400 for x in range(401)]
LF = [fonction_f(m,M,x) for x in LX]
plt.plot(LX,LF,"*")
plt.show()
```

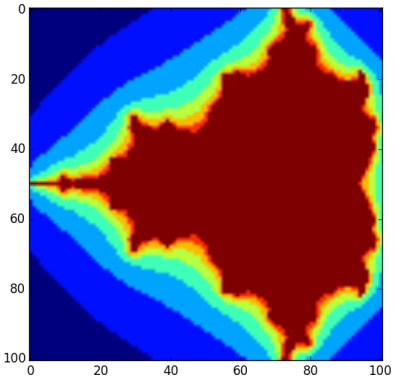



```
# Question 3
#===========
LX = [-2+2.5*x/100 for x in range(101)]
LY = [-1.1+2.2*x/100 for x in range(101)]
XY = [[[x,y] for x in LX] for y in LY]

for i in range(len(LX)):
    for j in range(len(LY)):
        XY[i][j]=fonction_f(
            m,M,complex(XY[i][j][0],XY[i][j][1]))
```

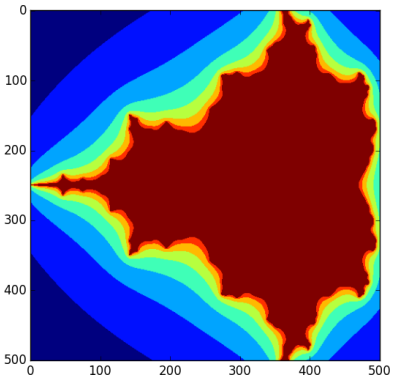
```
# Question 4
#===========
```

```
res = 100
LX = [-2+2.5*x/res for x in range(res+1)]
LY = [-1.1+2.2*x/res for x in range(res+1)]
XY = [[[x,y] for x in LX] for y in LY]

for i in range(len(LX)):
    for j in range(len(LY)):
        XY[i][j]=fonction_f(
        m,M,complex(XY[i][j][0],XY[i][j][1]))

# plt.imshow(XY)
# plt.show()
```

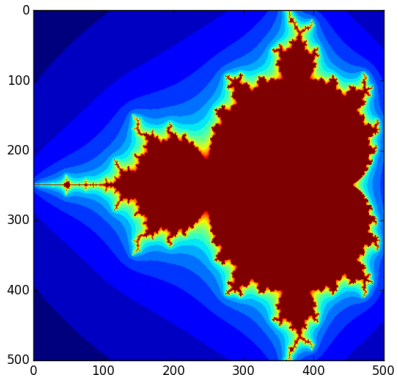
```
# Bilan
#=======
def affichage(m,M,res):
    m = m
    M = M
    LX = [-2+2.5*x/res for x in range(res+1)]
    LY = [-1.1+2.2*x/res for x in range(res+1)]
    XY = [[[x,y] for x in LX] for y in LY]
    for i in range(len(LX)):
        for j in range(len(LY)):
            XY[i][j]=fonction_f(
                m,M,complex(XY[i][j][0],XY[i][j][1]))
    plt.imshow(XY)
    plt.show()
```



$m = 10, M = 20$
100 points par 100 points



$m = 10, M = 20$
500 points par 500 points



$m = 20, M = 40$
500 points par 500 points

Exercice 10 – Corrigé

```
import numpy as np
# Question 1
R = np.array([[1,2,3],[4,5,6]])
S = np.array([[1,2,3],[4,5,6],[7,8,9]])
```

```
# Question 2
def test(M):
    """
    Fonction permettant de tester si la
    matrice est carrée et retournant sa taille.
    Entrée :
     * M(numpy.ndarray) : matrice
    Sortie :
     * 0 si taille non carrée
     * n(int) : taille de M si elle est carrée
```

```
    """
    l = M.shape[0]
    c = M.shape[1]
    if l==c :
        return l
    else :
        return 0
print(test(R),test(S))
```

```
# Question 3
fid = open("data/ex_006.txt",'r')
M1 = []
for ligne in fid :
    l = ligne.rstrip().split(" ")
    Ligne = [float(x) for x in l]
    M1.append(Ligne)
fid.close()
M1 = np.array(M1)
```

```
# Question 4
if test(M1)>0:
    valeurs_propres = np.linalg.eig(M1)[0]
    print(valeurs_propres)
```

```
# Question 5
def dansIntervalle(L,a,b):
    """
    Vérifier que chaque élément de L est dans
    l'intervalle [a,b]
    Entrées :
     * L(lst) : liste de nombres
     * a,b(flt) : nombres
    Sortie :
     * True si chaque élément est dans [a,b]
     * False sinon.
    """
    for e in L :
        if e<a or e> b:
            return False
    return True
print(dansIntervalle(valeurs_propres,0,1))
```

Exercice 11 – Tri de liste – Corrigé

```
# Question 1
# ==========
def comptage(L,n):
    """
    Comptage des éléments de L.
    Entrées :
     * n(int) : entier
     * L(lst) : liste d'éléments inférieurs àn
    """
    P = [0 for i in range(n+1)]
    # P = [0]*(n+1)
    for e in L:
        P[e]=P[e]+1
    return P
from random import randint
maxi = 5
LL = [randint(0,maxi) for x in range(20)]
```

```
P = comptage(LL,maxi)
# print(LL)
# print(P)
```

```
# Question 2
# ==========
def tri(L,n):
    """
    Tri une liste.
    Entrées :
     * n(int) : entier
     * L(lst) : liste d'éléments inférieurs àn
    Sortie :
     * T(lst) : liste triée.
    """
    P = comptage(L,n)
    T = []
    for i in range(len(P)):
        for j in range(P[i]):
            T.append(i)
    return T
```

```
# Question 3
# ==========
from random import randint
maxi = 5
LL = [randint(0,maxi) for x in range(20)]
T = tri(LL,maxi)
print(LL)
print(T)
```

```
# Question 4
# ==========
# Complexité quadratique : C(n)=O(n+n^2)=O(n^2)
# n : complexité de comptage
# n^2 : complexité des deux boucles imbriquées du
# tri
# Ce tri s'exécutera toujours dans le pire des cas.
# Dans le cas moyen : tri fusion O(nlogn)
# Dans le cas moyen : tri insertion O(n^2)
```

Exercice 12 – Corrigé

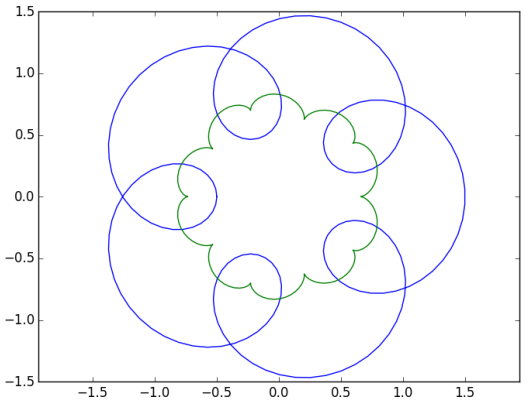
```
b,w = 0.5,6
# Question 1
# ==========
import numpy as np
def fonc_p(t):
    return [np.cos(t)+b*np.cos(w*t),np.sin(t)
                      +b*np.sin(w*t)]
```

```
def fonc_v(t):
    return [-np.sin(t)-b*w*np.sin(w*t),np.cos(t)
                      +b*w*np.cos(w*t)]
```

```
def fonc_a(t):
    return [-np.cos(t)-b*w*w*np.cos(w*t),
                -np.sin(t)-b*w*w*np.sin(w*t)]
```

```
# Question 2
# ==========
L=np.linspace(-np.pi,np.pi,200)
```

```
# Question 3
# ==========
import matplotlib.pyplot as plt
p = fonc_p(L)
#plt.plot(p[0],p[1])
#plt.axis("equal")
#plt.show()
```



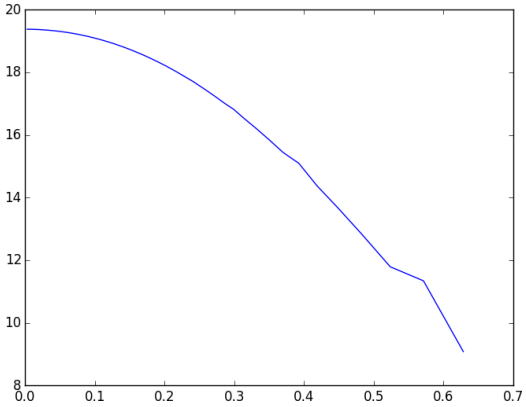

```
# Question 4
# ==========
def fonc_d(t):
    xp,yp = fonc_v(t)
    xpp,ypp = fonc_a(t)
    return (xp**2 + yp**2)/(xp*ypp-yp*xpp)
def fonc_c(t):
    fd = fonc_d(t)
    x,y = fonc_p(t)
    xp,yp = fonc_v(t)
    return [x-fd*yp,y+fd*xp]
```

```
# Question 5
# ==========
les_xc = []
les_yc = []
c = fonc_c(L)
#plt.plot(c[0],c[1])
```

```
# Question 6
# ==========
from math import sqrt
def distance(p):
    """
```

```
    Calcule la longueur du profil p.
    Entrée :
     * p(lst) : liste [les_x,les_y]
    Sortie :
     * L(flt) : longueur du profil.
    """
    L=0
    for i in range(len(p[0])-1):
        x0 = p[0][i]
        y0 = p[1][i]
        x1 = p[0][i+1]
        y1 = p[1][i+1]
        L = L+ sqrt((x1-x0)**2+(y1-y0)**2)
    return L
```

```
les_dt = []
les_dist = []
for i in range(10,2000,1) :
    dt = 2*np.pi/i
    L=np.linspace(-np.pi,np.pi,i)
    p = fonc_p(L)
    d = distance(p)
    les_dt.append(dt)
    les_dist.append(d)
plt.plot(les_dt,les_dist)
plt.show()
```



Évolution de la longueur du polynôme en fonction de δt.

6 Correction – Adaptés des exercices de F. Butin

Exercice 1 – Corrigé

```
# Question 1
# ==========
def affiche_poly(P):
    """
    Affiche un polynome sous la forme
    a0*X^0+a1*X^1+a2*X^2+...
    Entrée :
     * P(lst) : liste des coefficients du polynome
       [a0,a1,...,an]
    Sortie :
     * Rien. Affichage
    """
    ch=""
    for i in range(len(P)):
        signe="+"
        if P[i]<0 :
            signe="-"

        ch=ch+signe+str(abs(P[i]))+"*X^"+str(i)
    print(ch)
```

```
# Question 2
# ==========
def degre_poly(P):
    """
    Calcule le degré d'un polynome.
    On se base uniquement sur la taille de la liste
    à cause de la comparaison àzero
    Entrée :
     * P(lst) : liste des coefficients du polynome
       [a0,a1,...,an]
    Sortie :
     * deg(int) : dege du polynome
    """
    return len(P)-1
```

```
# Question 3
# ==========
def add_poly(P1,P2):
    """
    Calcule la somme de deux polynomes.
    Attention àbien faire une copie...
    Entrée :
     * P1, P2(lst) : liste des coefficients du
        polynome [a0,a1,...,an]
    Sortie :
     * P(lst) : liste des coefficients du
        polynome [a0+b0,a1+b1,...,an+bn]
    """
    # On cherche le polynome le plus grand
    if degre_poly(P1)>=degre_poly(P2):
        P=P1.copy()
        for i in range(len(P2)):
            P[i]=P[i]+P2[i]
    else :
        P=P2.copy()
        for i in range(len(P1)):
            P[i]=P[i]+P1[i]
    return P
```

```
def mul_poly(P1,P2):
    """
    Calcule la multiplication de deux polynomes.
    Entrée :
     * P1, P2(lst) : liste des coefficients du
       polynome [a0,a1,...,an]
    Sortie :
     * P(lst) : produits des polynomes
    """
    P=[0]*(degre_poly(P1)+degre_poly(P2)+1)
    for i in range(len(P1)):
        for j in range(len(P2)):
           P[i+j] = P[i+j]+ P1[i]*P2[j]
    return P
```


```
def mul_poly(P1,P2):
    """
    Calcule la multiplication de deux polynomes.
    Entrée :
     * P1, P2(lst) : liste des coefficients du
       polynome [a0,a1,...,an]
    Sortie :
     * P(lst) : produits des polynomes
    """
    P=[0]*(degre_poly(P1)+degre_poly(P2)+1)
    for i in range(len(P1)):
        for j in range(len(P2)):
           P[i+j] = P[i+j]+ P1[i]*P2[j]
    return P
```

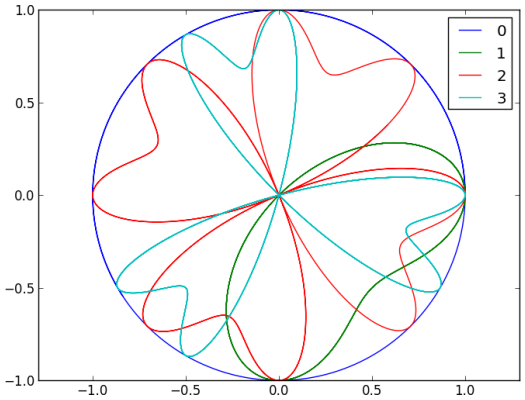

```
# Question 4
# ==========
def prsc_poly(P1,P2):
    """
    Calcule le produit scalaire de deux polynomes.
    Attention àbien faire une copie...
    Entrée :
     * P1, P2(lst) : liste des coefficients du
       polynome [a0,a1,...,an]
    Sortie :
     * P(flt) :produit des coefficients des
        polynomes a0*b0+a1*b1+...
    """
     # On cherche le polynome le plus grand
    if degre_poly(P1)>=degre_poly(P2):
        P=P1.copy()
        for i in range(len(P2)):
            P[i]=P[i]*P2[i]
    else :
        P=P2.copy()
        for i in range(len(P1)):
            P[i]=P[i]*P1[i]
    return sum(P)
```

```
# Question 5
# ==========
def deriv_poly(P):
    """
    Calcule la dérivée d'un polynome.
    Entrée :
     * P(lst) : liste des coefficients du
        polynome [a0,a1,...,an]
    Sortie :
```

```
    * coefficients du polynome dérivé
    """
    return [P[i]*i for i in range(1,len(P))]
```

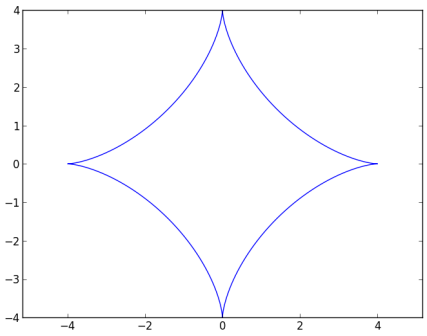
Exercice 3 – Corrigé

```
import numpy as np
import matplotlib.pyplot as plt
# Question 1
# ==========
def f_sigma(n,t):
    return (np.cos(n*t))**3-(np.sin(n*t))**3

x = np.linspace(0,10,1000)
y = f_sigma(1,x)

for i in range(4):
    x = np.linspace(0,10,1000)
    y = f_sigma(i,x)
    plt.plot(y*np.cos(x),y*np.sin(x),label=str(i))

plt.legend()
plt.axis("equal")
plt.show()
```

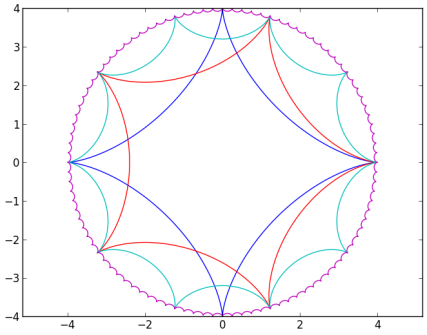


Exercice 7 – Corrigé

```
# Question 2
# ==========
R=4
r=R/4
m = 1-R/r
theta = np.linspace(0,2*np.pi,1000)
x=(R-r)*np.cos(theta)+r*np.cos(m*theta)
y=(R-r)*np.sin(theta)+r*np.sin(m*theta)
plt.plot(x,y,label="R=4, r=1, m=-3")
```



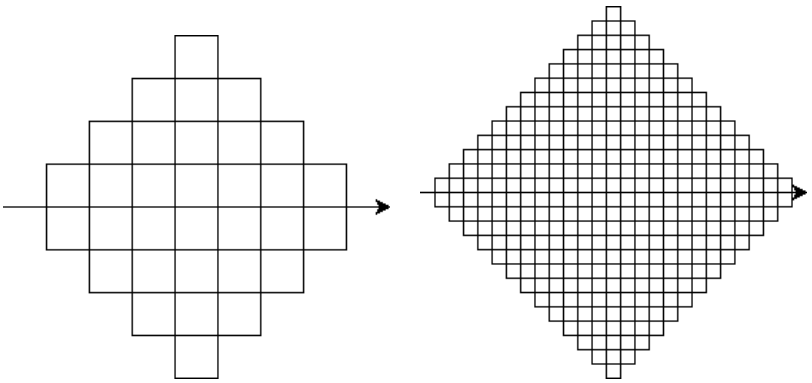
```
# Question 3
# ==========
for p in [1, 5, 10, 100]:
    r=R/p
    m = 1-R/r
    theta = np.linspace(0,2*np.pi,1000)
    x=(R-r)*np.cos(theta)+r*np.cos(m*theta)
    y=(R-r)*np.sin(theta)+r*np.sin(m*theta)
    titre = "R="+str(R)+", r = "+str(r)+ ",m="+str(m)
    plt.plot(x,y,label=titre)

x=R*np.cos(m*theta)
y=R*np.sin(m*theta)
#plt.plot(x,y)

plt.axis("equal")
#plt.legend()
plt.show()
```



Exercice 9 – Corrigé

```
# EXERCICE 9
import numpy as np
import matplotlib.pyplot as plt
import turtle

# Question 1
#============
turtle.hideturtle()
turtle.goto(100,0)
turtle.speed(1)
tortue = turtle.Pen()

t = np.linspace(0,2*np.pi,1000)
x=100*np.cos(t)
y=100*np.sin(t)
for i in range(len(x)):
```

```
    tortue.goto(x[i],y[i])
```



```
# Question 2
# ==========
def peano(n) :
    if n==1 :
        tortue.forward(10)
    else :
        peano(n-1)
        tortue.left(90)
        peano(n-1)
        tortue.left(-90)
        peano(n-1)
        tortue.left(-90)
        peano(n-1)
        tortue.left(-90)
        peano(n-1)
        tortue.left(90)
        peano(n-1)
        tortue.left(90)
        peano(n-1)
        tortue.left(90)
        peano(n-1)
        tortue.left(-90)
        peano(n-1)

peano(4)
turtle.hideturtle()
```

Exercice 10 – Corrigé

```
# EXERCICE 10 FB
import numpy as np
import matplotlib.pyplot as plt
import turtle
__author__ = "Frederic Butin"

# Question 1
# ==========
turtle.hideturtle()
turtle.speed(10);
tortue = turtle.Pen()

def koch (n):
    if n==1 :
        tortue.forward(3)
    else :
        koch(n-1);
        tortue.left(60)
        koch(n-1);
        tortue.left(-120)
```

```
        koch(n-1);
        tortue.left(60)
        koch(n-1)


def flocon(n):
    tortue.clear()
    koch(n)
    tortue.left(-120)
    koch(n)
    tortue.left(-120)
    koch(n)
    tortue.hideturtle()

flocon(5)
```

Exercice 11 – Corrigé

```
# EXERCICE 11
# Question 1
# ==========
import matplotlib.pyplot as plt
import math

I_th = 8*math.log(2)-3
```

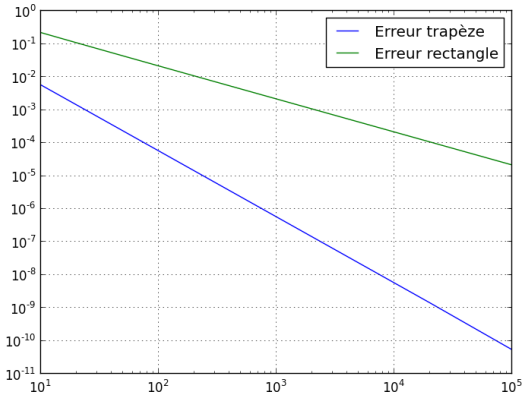
```
# Question 2
# ==========
def fonc(x):
    return math.log(x)
```

```
def calc_int_trap(a,b,n):
    res = 0
    pas = (b-a)/n
    x = a+pas
    i=1
    while i<n:
        res = res + fonc(x)
        x = x + pas
        i=i+1
    res = pas*(res+(fonc(a)+fonc(b))/2)
    return res
```

```
def calc_int_rect_g(a,b,n):
    res = 0
    pas = (b-a)/n
    x = a
    i=0
    while i<n:
        res = res + fonc(x)
        x = x + pas
        i=i+1
    return res*pas
```

```
N = [10,20,40,100,200, 400, 500, 600, 700, 800,
       900, 1000, 5000, 10000, 20000, 100000]

calc_int_trap(1,4,10)
```

```
err_trap = [abs(I_th-calc_int_trap(1,4,n))
                                    for n in N]
err_rect = [abs(I_th-calc_int_rect_g(1,4,n))
                                    for n in N]

plt.loglog(N,err_trap,label = "Erreur trapèze")
plt.loglog(N,err_rect,label = "Erreur rectangle")
plt.legend()
plt.grid()
plt.show()
```



Exercice 12 – Corrigé

A refaire ?

On pose :

$$\begin{cases} y_1(t) = x(t) \\ y_2(t) = x'(t) \end{cases}$$

En utilisant le schéma d'Euler explicite, on a :

$$\begin{cases} y_1'(k) = \dfrac{y_1(k+1) - y_1(k)}{h} \\ y_2'(k) = \dfrac{y_2(k+1) - y_2(k)}{h} \end{cases}$$

On a donc :

$$\begin{cases} y_1'(t) = y_2(t) \\ y_2'(t) + 10y_2(t) - y_1(t) = \sin(nt) \end{cases}$$

En discrétisant l'équation on a donc :

$$\begin{cases} y_2(k) = \dfrac{y_1(k+1) - y_1(k)}{h} \\ \dfrac{y_2(k+1) - y_2(k)}{h} + 10y_2(k) - y_1(k) = \sin(nk) \end{cases}$$

$$\Leftrightarrow \begin{cases} y_1(k+1) = h\,y_2(k) + y_1(k) \\ y_2(k+1) = h\big(\sin(nk) + y_1(k) - 10y_2(k)\big) + y_2(k) \end{cases}$$

Mise en forme matricielle du problème : On pose :

$$X = \begin{bmatrix} x(t) \\ x'(t) \end{bmatrix} \Rightarrow X' = \begin{bmatrix} x'(t) \\ x''(t) \end{bmatrix}$$

On a alors :

$$\begin{bmatrix} x'(t) \\ x''(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & -10 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ x'(t) \end{bmatrix} + \begin{bmatrix} 0 \\ \sin(nt) \end{bmatrix}$$

Le système peut donc se mettre sous la forme :

$$X'(t) = AX(t) + B(t)$$

En appliquant un schéma d'Euler explicite, on a donc :

$$X'(t) \simeq \frac{X_{k+1} - X_k}{h}$$

D'où :

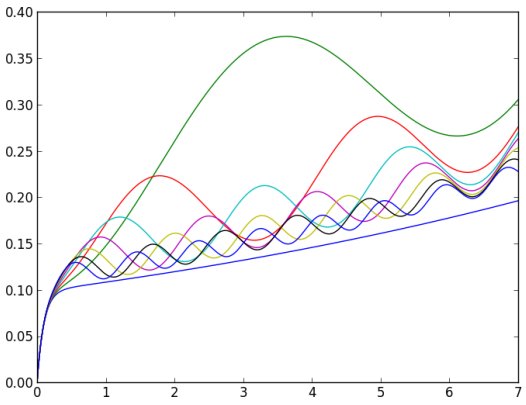
$$\frac{X_{k+1} - X_k}{h} = AX_k + B_k \Leftrightarrow X_{k+1} = (hA+1)X_k + hB_k$$

Mise en forme du problème de Cauchy :

En réutilisant la mise en forme matricielle précédente, on peut donc définit la fonction f telle que :

$$f(X,t) \mapsto AX(t) + B(t)$$

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.integrate as spi
N=10000

def fonction_f(X,t,n):
   return [X[1],X[0]-10*X[1]+np.sin(n*t)]

les_t=np.linspace(0,7,N)
for i in range(0,11):
    res = spi.odeint(fonction_f,[0,1],les_t,(i,))
    plt.plot(les_t,res[:,0])
plt.show()
```



Exercice 13 – Corrigé

On pose :

$$X_i = \begin{bmatrix} F_{i+1} \\ F_i \end{bmatrix}$$

Le problème peut être mis sous la forme matricielle suivante :

$$X_{i+1} = \begin{bmatrix} F_{i+2} \\ F_{i+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{i+1} \\ F_i \end{bmatrix} \Leftrightarrow X_{i+1} = MX_i$$

On peut donc écrire que :

$$X_{i+1} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_{i+1} \\ F_i \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F_i \\ F_{i-1} \end{bmatrix}$$

On a donc :

$$X_n = M^n X_0$$

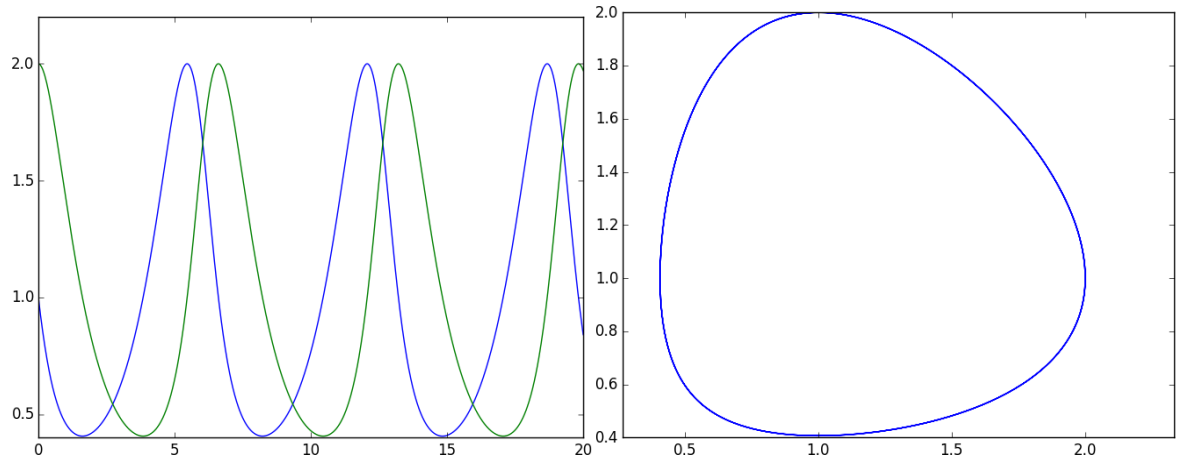
Exercice 19

```
# EXERCICE 19
# Question 1
# ==========
import numpy as np
import scipy.integrate as sci
import matplotlib.pyplot as plt

def fonction_PP(X,t,a,b,c,d):
    return [a*X[0]-b*X[0]*X[1],-c*X[1]+d*X[0]*X[1]]

def fonction_PP2(X,t):
    a,b,c,d = 1,1,1,1
    return [a*X[0]-b*X[0]*X[1],-c*X[1]+d*X[0]*X[1]]
```

```
# Question 2
# ==========
# Conditions initiale :
X0=[1,2]
t = np.linspace(0,20,1000)
res = sci.odeint(fonction_PP2,X0,t)
```

```
# Question 3
# ==========
plt.figure()
plt.plot(t,res[:,0])
plt.plot(t,res[:,1])
```

```
# Question 4
# ==========
plt.figure()
plt.plot(res[:,0],res[:,1])
plt.axis("equal")
```



Mémento Python 3

Types de base

entier, flottant, complexe, booléen, chaîne

```
  int   783     0     -192
float   9.23    0.0   -1.7e-6
complex 2.7+3.1j  1j
 bool   True    False
  str   "Un\nDeux"   'L\'âme'
```

- `\n` : retour à la ligne
- `\'` : ' échappé
- multiligne : `"""X\tY\tZ` / `1\t2\t3"""` (`\t` : tabulation)
- `-1.7e-6` : 10^{-6}
- *non modifiable, séquence ordonnée de caractères*

Types Conteneurs (listes, tuples, chaînes)

- séquences ordonnées, accès index rapide, valeurs répétables

```
  list  [1,5,9]   ["x",11,8.9]   ["mot"]   []
 tuple  (1,5,9)   11,"y",7.4     ("mot",)  ()
```

non modifiable — expression juste avec des virgules

`str` en tant que séquence ordonnée de caractères

Identificateurs

pour noms de variables, fonctions, modules, classes…

`a..zA..Z_` suivi de `a..zA..Z_0..9`

- accents possibles mais à éviter
- mots clés du langage interdits
- distinction casse min/MAJ

☺ `a toto x7 y_max BigOne`
☹ ~~`8y and`~~

Affectation de variables

```
x = 1.2+8+sin(0)
y,z,r = 9.2,-7.6,"bad"
x+=3
x-=2
x=None
```

- `x` : nom de variable (identificateur)
- `1.2+8+sin(0)` : valeur ou expression de calcul
- `y,z,r` : noms de variables
- `9.2,-7.6,"bad"` : conteneur de plusieurs valeurs (ici un tuple)
- `x+=3` / `x-=2` : incrémentation / décrémentation
- `x=None` : valeur constante « non défini »

Conversions

`type(`*expression*`)`

```
int("15")
int(15.56)
float("-11.24e8")
str(78.3)
```

`int("15")` : on peut spécifier la base du nombre entier en 2nd paramètre

`int(15.56)` : troncature de la partie décimale (`round(15.56)` pour entier arrondi)

`str(78.3)` : et pour avoir la représentation littérale → `repr("Texte")`

voir au verso le formatage de chaînes, qui permet un contrôle fin

`bool` → utiliser des comparateurs (avec `==`, `!=`, `<`, `>`, …), résultat logique booléen

`list("abc")` → *utilise chaque élément de la séquence en paramètre* → `['a','b','c']`

`":".join(['toto','12','pswd'])` → `'toto:12:pswd'`

- `":"` : chaîne de jointure
- `['toto','12','pswd']` : séquence de chaînes

`"Un blanc final \n".strip()` → `"Un blanc final"`

`"des mots  espacés".split()` → `['des','mots','espacés']`

`"1,4,8,2".split(",")` → `['1','4','8','2']`

- `","` : chaîne de séparation

Indexation des listes, tuples, chaînes de caractères...

```
index négatif   -6   -5   -4    -3   -2   -1
index positif    0    1    2     3    4    5
lst=           [11,  67,  "abc", 3.14, 42, 1968]
tranche positive 0    1    2     3    4    5    6
tranche négative -6   -5   -4    -3   -2   -1
```

`len(lst)` → `6`

accès individuel aux éléments par `[`*index*`]`

- `lst[1]` → `67`
- `lst[0]` → `11` *le premier*
- `lst[-2]` → `42`
- `lst[-1]` → `1968` *le dernier*

accès à des sous-séquences par `[`*tranche début*`:`*tranche fin*`:`*pas*`]`

- `lst[:-1]` → `[11,67,"abc",3.14,42]`
- `lst[1:-1]` → `[67,"abc",3.14,42]`
- `lst[::2]` → `[11,"abc",42]`
- `lst[:]` → `[11,67,"abc",3.14,42,1968]`
- `lst[1:3]` → `[67,"abc"]`
- `lst[-3:-1]` → `[3.14,42]`
- `lst[:3]` → `[11,67,"abc"]`
- `lst[4:]` → `[42,1968]`

Indication de tranche manquante → à partir du début / jusqu'à la fin.

Sur les séquences modifiables, utilisable pour suppression `del lst[3:5]` *et modification par affectation* `lst[1:4]=['hop',9]`

Logique booléenne

Comparateurs: `<` `>` `<=` `>=` `==` `!=` (≤ ≥ = ≠)

- `a and b` et logique — *les deux en même temps*
- `a or b` ou logique — *l'un ou l'autre ou les deux*
- `not a` non logique
- `True` valeur constante vrai
- `False` valeur constante faux

Blocs d'instructions

instruction parente :
indentation !
bloc d'instructions 1…
⋮
instruction parente :
bloc d'instructions 2…
⋮
instruction suivante après bloc 1

Instruction conditionnelle

bloc d'instructions exécuté uniquement si une condition est vraie

`if` *expression logique* `:`
→ *bloc d'instructions*

combinable avec des sinon si, sinon si... et un seul sinon final.

```
if x==42:
    # bloc si expression logique x==42 vraie
    print("vérité vraie")
elif x>0:
    # bloc sinon si expression logique x>0 vraie
    print("positivons")
else:
    # bloc sinon des autres cas restants
    print("ça veut pas")
```

Maths

nombres flottants… valeurs approchées ! — *angles en radians*

Opérateurs: `+` `-` `*` `/` `**` (× ÷ a^b)

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
acos(0.5)→1.0471…
sqrt(81)→9.0
log(e**2)→2.0
```

√ *etc. (cf doc)*

Complexes

```
z=1+2j
z.real
z.imag
z.conjugate()
abs(z)
```

Opérations spécifiques aux entiers

`17 % 5` reste et
`17 // 5` quotient
dans la div. eucl. de 17 par 5

Instruction boucle conditionnelle

bloc d'instructions exécuté tant que la condition est vraie

```
while expression logique:
    bloc d'instructions
```

```
s = 0
i = 1
```
*initialisations **avant** la boucle*

condition avec au moins une valeur variable (ici `i`)

```
while i <= 100:
    # bloc exécuté tant que i ≤ 100
    s = s + i**2
    i = i + 1
```
faire varier la variable de condition !

$$s = \sum_{i=1}^{i=100} i^2$$

```
print("somme:",s)
```
résultat de calcul après la boucle

attention aux boucles sans fin !

contrôle de boucle :

`break` sortie immédiate `continue` itération suivante

Instruction boucle itérative

bloc d'instructions exécuté pour chaque élément d'un conteneur ou d'un itérateur

```
for variable in séquence:
    bloc d'instructions
```

Parcours des **valeurs** de la séquence

```
s = "Du texte"
cpt = 0
```
*initialisations **avant** la boucle*

variable de boucle, valeur gérée par l'instruction `for`

```
for c in s:
    if c == "e":
        cpt = cpt + 1
print("trouvé",cpt,"'e'")
```
Comptage du nombre de `e` dans la chaîne.

boucle sur dict/set = boucle sur séquence des clés

utilisation des tranches pour parcourir un sous-ensemble de la séquence

Parcours des **index** de la séquence

- changement de l'élément à la position
- accès aux éléments autour de la position (avant/après)

```
lst = [11,18,9,12,23,4,17]
perdu = []
for idx in range(len(lst)):
    val = lst[idx]
    if val> 15:
        perdu.append(val)
        lst[idx] = 15
print("modif:",lst,"-modif:",perdu)
```
Bornage des valeurs supérieures à 15, mémorisation des valeurs perdues.

Parcours simultané **index** et **valeur** de la séquence:

```
for idx,val in enumerate(lst):
```

Affichage / Saisie

```
print("v=",3,"cm :",x,",",y+4)
```

éléments à afficher : valeurs littérales, variables, expressions

Options de `print`:

- `sep=" "` (séparateur d'éléments, défaut espace)
- `end="\n"` (fin d'affichage, défaut fin de ligne)
- `file=f` (print vers fichier, défaut sortie standard)

```
s = input("Directives:")
```

`input` retourne toujours une **chaîne**, la convertir vers le type désiré (cf encadré *Conversions* au recto).

Génération de séquences d'entiers

très utilisé pour les boucles itératives for

`range([début,] fin [,pas])` — par défaut 0, non compris

```
range(5)      → 0 1 2 3 4
range(3,8)    → 3 4 5 6 7
range(2,12,3) → 2 5 8 11
```

`range` retourne un « générateur », faire une conversion en liste pour voir les valeurs, par exemple:

```
print(list(range(4)))
```

Opérations sur conteneurs (listes, tuples, chaînes)

```
len(c) → nb d'éléments
min(c)  max(c)  sum(c)
sorted(c) → copie triée
val in c → booléen, opérateur in de test de présence (not in d'absence)
enumerate(c) → itérateur sur (index,valeur)
```

*Spécifique aux **conteneurs de séquences** (listes, tuples, chaînes) :*

```
reversed(c) → itérateur inversé   c*5 → duplication   c+c2 → concaténation
c.index(val) → position           c.count(val) → nb d'occurences
```

Définition de fonction

nom de la fonction (identificateur) — paramètres nommés

```
def nomfct(p_x,p_y,p_z):
    """documentation"""
    # bloc instructions, calcul de res, etc.
    return res
```
valeur résultat de l'appel. si pas de résultat calculé à retourner : `return None`

les paramètres et toutes les variables de ce bloc n'existent que *dans* le bloc et *pendant* l'appel à la fonction (« *boite noire* »)

Appel de fonction

```
r = nomfct(3,i+2,2*i)
```
un argument par paramètre

récupération du résultat renvoyé (si nécessaire)

Opérations spécifiques aux listes

modification de la liste originale

`lst.append(item)`	ajout d'un élément à la fin
`lst.extend(seq)`	ajout d'une séquence d'éléments à la fin
`lst.insert(idx,val)`	insertion d'un élément à une position
`lst.remove(val)`	suppression d'un élément à partir de sa valeur
`lst.pop(idx)`	suppression de l'élément à une position et retour de la valeur
`lst.sort()` `lst.reverse()`	tri / inversion de la liste *sur place*

Listes par compréhension

```
lst = [2*i for i in range(10)]
lst = [i for i in range(20) if i%2 == 0]
```
lst = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Fichiers

stockage de données sur disque, et relecture

```
f = open("fic.txt","r",encoding="utf8")
```

variable fichier pour les opérations

nom du fichier sur le disque, chemin, relatif ou absolu

mode d'ouverture
- `'r'` lecture (read)
- `'w'` écriture (write)
- `'a'` ajout (append)…

encodage des caractères pour les fichiers textes: `utf8` `ascii` `latin1` …

en **écriture**

```
f.write("coucou")
```
*fichier texte → lecture / écriture de **chaînes** uniquement, convertir de/vers le type désiré*

en **lecture**

```
s = f.read(4)
```
chaîne vide si fin de fichier; si nb de caractères pas précisé, lit tout le fichier

```
s = f.readline()
```
lecture ligne suivante

```
f.close()
```
ne pas oublier de refermer le fichier après son utilisation !

très courant : boucle itérative de lecture des lignes d'un fichier texte :

```
for ligne in f :
    bloc de traitement de la ligne
```

Ce mémento est fourni à titre indicatif. Il ne faut le considérer :

- *ni comme exhaustif (en cas de problème sur un exercice particulier, si une fonction ou une commande indispensable était absente de la liste, l'interrogateur pourrait aider le candidat),*
- *ni comme exclusif (une fonction ou une commande absente de cette liste n'est pas interdite : si un candidat utilise à très bon escient d'autres fonctions MAIS sait aussi répondre aux questions sur les fonctions de base, il n'y a pas de problème),*
- *ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).*

Les fonctions et commandes présentées doivent simplement permettre de faire les exercices proposés aux candidats.

L'examinateur n'attend pas du candidat une connaissance encyclopédique du langage Python, mais une utilisation raisonnée des principes algorithmiques et une mise en pratique des connaissances de base.

L'utilisation de l'aide en ligne est encouragée, mais ne doit pas masquer une ignorance sur ces aptitudes.

Aide

`help(a)` → aide sur a `dir(a)` → liste d'attributs de a `F1`

`help("module.obj")` → aide sur obj de module, sans avoir besoin d'importer le module

Mémento numérique Python 3

`import matplotlib.pyplot as plt` → charge le module pyplot sous le nom `plt`

`plt.figure('titre')` → crée une fenêtre de tracé vide

`plt.plot(LX,LY,'o-b')` → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)
- couleur : `'b'` (**blue**), `'g'` (**green**), `'r'` (**red**), `'c'` (cyan), `'m'` (**magenta**), `'y'` (yellow), `'k'` (**black**)
- type de ligne : `'-'` (trait plein), `'--'` (pointillé), `'-.'` (alterné)...
- marque : `'o'` (rond), `'h'` (hexagone), `'+'` (plus), `'x'` (croix), `'*'` (étoile)...

`plt.xlim(xmin,xmax)` → fixe les bornes de l'axe x

`plt.ylim(ymin,ymax)` → fixe les bornes de l'axe y

`plt.axis('equal')` → change les limites des axes x et y pour un affichage avec des axes orthonormés (le tracé d'un cercle donne un cercle)

`plt.show()` → affichage de la fenêtre

`plt.savefig(fichier)` → sauve le tracé dans un fichier (le suffixe du nom fichier peut donner le format ; par exemple, `'image.png'`)

`import numpy as np` → charge le module numpy sous le nom `np`

Construction de tableaux (de type ndarray)

`np.zeros(n)` → crée un vecteur dont les n composantes sont nulles

`np.zeros((n,m))` → crée une matrice $n \times m$, dont les éléments sont nuls

`np.eye(n)` → crée la matrice identité d'ordre n

`np.linspace(a,b,n)` → crée un vecteur de n valeurs régulièrement espacées de a à b

`np.arange(a,b,dx)` → crée un vecteur de valeurs de a incluse à b exclue avec un pas dx

`M.shape` → tuple donnant les dimensions de M

`M.size` → le nombre d'éléments de M

`M.ndim` → le nombre de dimensions de M

`M.sum()` → somme de tous les éléments de M

`M.min()` → plus petit élément de M

`M.max()` → plus grand élément de M

argument **axis** optionnel : 0 → lignes, 1 → colonnes :

`M.sum(0)` → somme des lignes

`M.min(0)` → plus petits éléments, sur chaque colonne

`M.max(1)` → plus grands éléments, sur chaque ligne

`import numpy.linalg as la`

`la.det(M)` → déterminant de la matrice carrée M

`la.inv(M)` → inverse de M

`la.eigvals(M)` → valeurs propres de M

`la.matrix_rank(M)` → rang de M

`la.matrix_power(M,n)` → M^n (n entier)

`la.solve(A,B)` → renvoie X tel que $A\,X = B$

`import scipy.integrate as spi`

`spi.odeint(F,Y0,LT)`
→ renvoie une solution numérique du problème de Cauchy $\mathbf{Y}'(t) = \mathbf{F}(\mathbf{Y}(t),t)$, où $\mathbf{Y}$ est un vecteur d'ordre n, avec la condition initiale $\mathbf{Y}(t_0) =$ `Y0`, pour les valeurs de t dans la liste `LT` de longueur k commençant par t_0, sous forme d'une matrice $n \times k$

`spi.quad(f,a,b)` → renvoie une évaluation numérique de l'intégrale : $\int_a^b f(t)\,dt$

Conversion ndarray <-> liste

`V = np.array([1,2,3])` → V : vecteur (1 2 3)

`L = V.tolist()` → L : liste [1, 2, 3]

`M = np.array([[1,2],[3,4]])` → M : matrice $\begin{pmatrix}1 & 2\\ 3 & 4\end{pmatrix}$

`L = M.tolist()` → L : liste [[1, 2], [3,4]]

Extraction d'une partie de matrice

`M[i], M[i,:]` → ligne de M d'index i

`M[:,j]` → colonne de M d'index j

`M[i:i+h,j:j+l]` → sous-matrice $h \times l$

Copier un tableau avec la méthode copy :

`M2 = M1.copy()`

`M1+M2, M1*M2, M**2` → opérations « terme-à-terme »

`c*M` → multiplication de la matrice M par le scalaire c

`M+c` → matrice obtenue en ajoutant le scalaire c à chaque terme de M

`V1.dot(V2)` / `np.dot(V1,V2)` → renvoie le produit scalaire de deux vecteurs

`M.dot(V)` / `np.dot(M,V)` → renvoie le produit d'une matrice par un vecteur

`M1.dot(M2)` / `np.dot(M1,M2)` → renvoie le produit de deux matrices

`M.transpose()` / `np.transpose(M)` → renvoie une copie de M transposée (ne modifie pas M)

`M.trace()` / `np.trace(M)` → renvoie la trace de M

Fonctions mathématiques usuelles

`np.exp, np.sin, np.cos, np.sqrt` *etc.*

→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).

Mémento numérique Scilab

Définition d'une fonction

nom de la fonction — paramètres nommés

```
function y = nomfct(x1,x2,x3)
  ...
  ...
  y = ...
endfunction
```

Boucles

```
for x=0:0.1:1
   ...
end
```

```
while x>0 & x<1
   ...
end
```

Booléens `%T,%F` `&`(and) `|`(or)

Nombres remarquables `%pi,%e`

Instruction conditionnelle

```
if x==2 then
    disp("x vaut 2")
elseif x>=0 then
    disp("x est positif ou nul")
else
    disp("x est négatif")
end
```

Complexes

```
z=1+2*%i
real(z)
imag(z)
conj(z)
abs(z)
```

Graphiques

`figure(`*no*`,"BackgroundColor",[`*r*`,`*g*`,`*b*`],"Position",[`*bg*`,`*bh*`,`*larg*`,`*haut*`])`
→ crée une nouvelle fenêtre de tracé vide, avec le numéro *no* ; on peut régler (options) la couleur du fond et la position de la fenêtre dans l'écran (bord gauche, bord haut, largeur, hauteur).

`plot(LX,LY,'o-b')` → trace le graphique défini par les listes LX et LY (abscisses et ordonnées)
- couleur : `'b'` (blue), `'g'` (green), `'r'` (red), `'c'` (cyan), `'m'` (magenta), `'y'` (yellow), `'k'` (black)
- type de ligne : `'-'` (trait plein), `'--'` (pointillé), `'-.'` (alterné)...
- marque : `'o'` (rond), `'s'` (carré), `'+'` (plus), `'x'` (croix), `'*'` (étoile)...

`mlb_axis("equal",[`x_{min}`,`x_{max}`,`y_{min}`,`y_{max}`])` → repère orthonormé, plage d'affichage ; on peut donner les deux arguments ou seulement l'un des deux.

`xs2png(`*no*`,"mon_image.png")` → sauve au format PNG le tracé de la figure *no* dans le fichier nommé « `mon_image.png` »
Voir aussi : **`xs2bmp`** (BMP), **`xs2gif`** (GIF), **`xs2jpg`** (JPG), **`xs2eps`** (EPS), **`xs2pdf`** (PDF), **`xs2svg`** (SVG)

Calcul matriciel et vectoriel

Construction de tableaux

`zeros(1,n)` → crée un vecteur-ligne dont les *n* composantes sont nulles
`zeros(n,m)` → crée une matrice *n*×*m*, dont les éléments sont nuls
`eye(n,n)` → crée la matrice identité d'ordre *n*
`linspace(a,b,n)` → crée un vecteur-ligne de *n* valeurs régulièrement espacées de *a* à *b*
`a:dx:b` → crée un vecteur-ligne de valeurs de *a* incluse à *b* incluse avec un pas *dx*

Définition d'un vecteur, d'une matrice

`U = [1,2,3]` → *U* : liste = vecteur-ligne (1 2 3)
`V = [1;2;3]` → *V* : vecteur-colonne $\begin{pmatrix}1\\2\\3\end{pmatrix}$
`M = [1,2;3,4]` → *M* : matrice $\begin{pmatrix}1&2\\3&4\end{pmatrix}$

`size(M)` → vecteur-ligne des dimensions de *M*
`length(M)` → le nombre d'éléments de *M*

`sum(M)` → somme de tous les éléments de *M*
`min(M)` → plus petit élément de *M*
`max(M)` → plus grand élément de *M*
deuxième argument optionnel :
"r"→ sur chaque colonne, renvoie un vecteur-ligne
"c"→ sur chaque ligne, renvoie un vecteur-colonne
`sum(M,"r")`→ somme des lignes
`min(M,"c")`→ plus petits éléments, de chaque ligne
`max(M,"r")`→ plus grands, de chaque colonne

Extraction d'une partie de matrice

`M(i,:)` **`M(i,1:$)`** → ligne de *M* d'index *i* (vecteur-ligne)
`M(:,j)` **`M(1:$,j)`** → colonne de *M* d'index *j* (vecteur-colonne)
`M(i:i+h,j:j+l)` → sous-matrice $(h+1)\times(l+1)$

Remarque sur les fonctions mathématiques usuelles

`exp, sin, cos, sqrt` *etc.*
→ fonctions qui s'appliquent sur des réels ou des complexes, mais aussi sur des vecteurs et des matrices (s'appliquent à chaque terme), qui sont optimisées en durée de calcul.

Algèbre linéaire

`trace(M)` → trace de la matrice carrée *M*
`det(M)` → déterminant de la matrice carrée *M*
`inv(M)` → inverse de *M*
`spec(M)` → valeurs propres de *M*
`rank(M)` → rang de *M*
`M^n` → M^n
`linsolve(A,B)` → renvoie *X* tel que *A X* = *B*

`M1+M2` → somme des deux matrices
`c*M` → multiplication d'une matrice *M* par un scalaire *c*
`M1.*M2, M.^2` → opérations « terme-à-terme »
`M+c` → matrice obtenue en ajoutant le scalaire *c* à chaque terme de *M*
`M'` → transposée de la matrice *M*
`V'` → passage d'un vecteur-ligne à un vecteur-colonne, et inversement
`M1*M2` → produit (matriciel) des deux matrices
`M*V` → produit d'une matrice par un vecteur-colonne
`V1'*V2` → produit scalaire de deux vecteurs-colonnes

Intégration numérique

`ode(Y0,t0,LT,F)`
→ renvoie une solution numérique du problème de Cauchy $\mathbf{Y}'(t)=\mathbf{F}(t,\mathbf{Y}(t))$, où $\mathbf{Y}$ est un vecteur d'ordre *n*, avec la condition initiale $\mathbf{Y}(t_0)$ = `Y0`, pour les valeurs de *t* dans la liste `LT` de longueur *k*, sous forme d'une matrice *n*×*k*

`intg(a,b,f)` → renvoie une évaluation numérique de l'intégrale : $\int_a^b f(t)\,dt$

Rappel : ce mémento est fourni à titre indicatif. Il ne faut le considérer ni comme exhaustif, ni comme exclusif, ni comme un minimum à connaître absolument (l'examinateur n'attend pas du candidat qu'il connaisse parfaitement toutes ces fonctions et ces commandes).