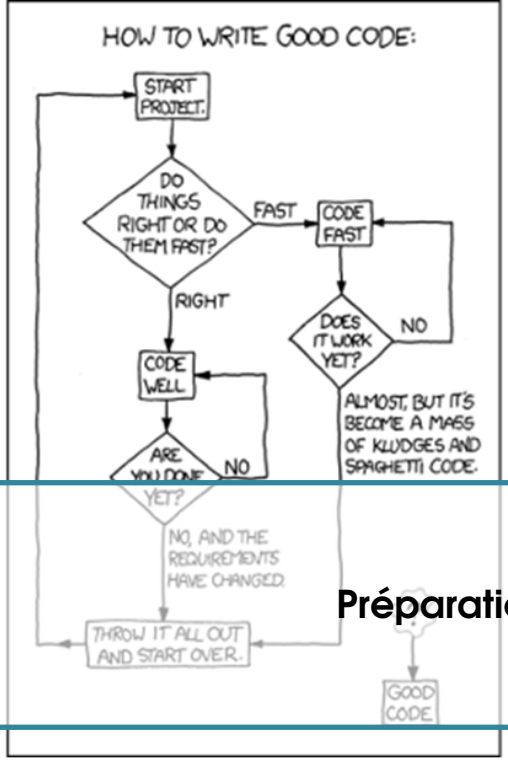
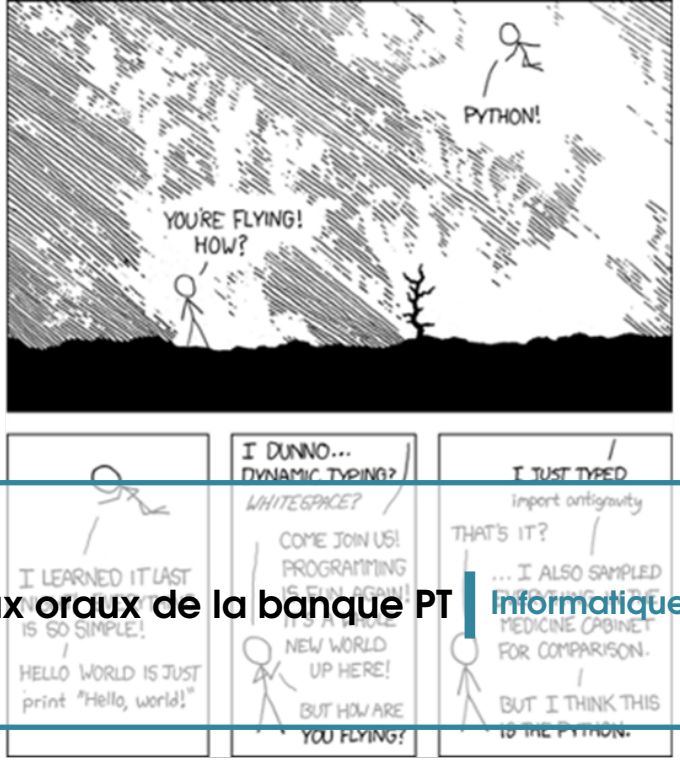





PT – PT ⋆

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 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
```

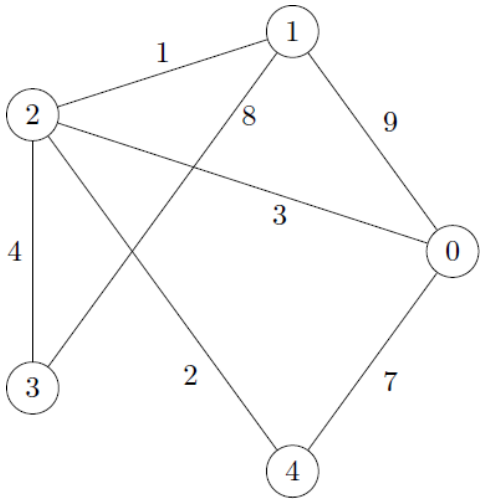
   Chaque ligne contient deux valeurs flottantes séparées par un point-virgule, représentant respectivement l'abscisse et l'ordonnée d'un point. Les points sont ordonnés par abscisses croissantes. Ouvrir le fichier en lecture, le lire et construire la liste `LX` des abscisses et la liste `LY` des ordonnées 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 $\pi$ avec un pas de discrétisation $\delta t$ 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 $\delta t$. Observer l'évolution de cette longueur lorsque $\delta t$ diminue.

## 2 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 $\Gamma_n$ en coordonnées polaires définie par :

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

1. Représenter la courbe $\Gamma_0$.
2. Représenter sur un même graphique les courbes $\Gamma_j$, 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 :

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

Oral

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 $f(t)$.

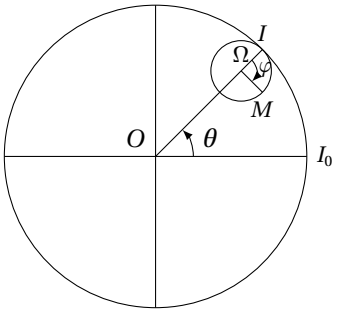
Quand elle existe, cette courbe est appelée l'enveloppe de la famille de droits $(D_t)_{t\in I}$.

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 $\Gamma$ dont étudie la trajectoire. On note $\theta$ l'angle $\left(\widehat{\overrightarrow{i}, \overrightarrow{O\Omega}}\right)$ et $\varphi$ l'angle $\left(\widehat{\overrightarrow{\Omega I}, \overrightarrow{\Omega M}}\right)$. Initialement, $\Omega$ 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 $m = 1 - \frac{R}{r}$. 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 $r = \frac{R}{4}$. Représenter la trajectoire de $M$. La courbe obtenue est appelée astroïde.
3. On choisit $R = 4$ et $r = \frac{R}{p}$ 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 $\frac{1}{3}$.

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()` ;
- 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 :

$$\begin{aligned} P_{11} &= M_{11}N_{11} + M_{12}N_{21} & P_{12} &= M_{11}N_{12} + M_{12}N_{22} \\ P_{21} &= M_{21}N_{11} + M_{22}N_{21} & P_{22} &= M_{21}N_{12} + M_{12}N_{22}. \end{aligned}$$

   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 $z\left(x_i,y_j\right)=0$ si $i\in\{0,n+1\}$ ou $j\in\{0,n+1\}$ par hypothèse. On note $z_{i,j}$ l'approximation cherchée de $z\left(x_i,y_j\right)$ par $z_{i,j}$ et le laplacien $\Delta z\left(x_i,y_j\right)$ 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 $A=\left(a_{i,j}\right)_{(i,j)\in[\![1,n]\!]^2}$ et $B=\left(b_{i,j}\right)_{(i,j)\in[\![1,n]\!]^2}$ 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 :

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

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 $\varphi$.
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.
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 $2\pi$-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.

# 3 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))
```

Oral

## 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

```
# 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))
```

```
# Question 3 et 4
# ================
```

## Exercice 5 – Corrigé

## 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 à résoudre 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...
```

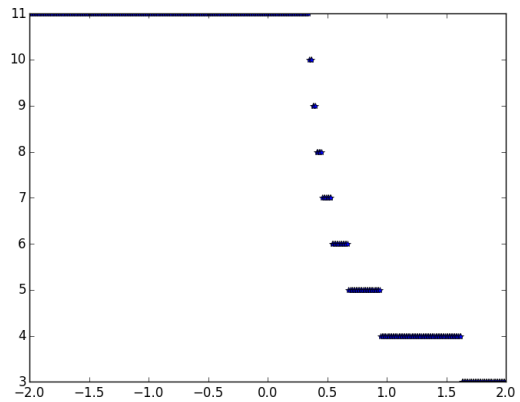
## Exercice 9 – Fractale de Mandelbrot – Corrigé

```
# 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
```

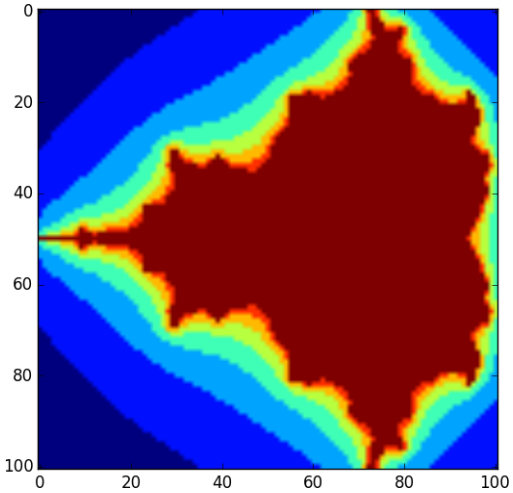
```
# Question 2
#===========
import matplotlib.pyplot as plt
m,M=10,20
LX = [-2+4*x/400 for x in range(401)]
LF = [fonction_f(m,M,x) for x in LX]
plt.plot(LX,LF,"*")
plt.show()
```



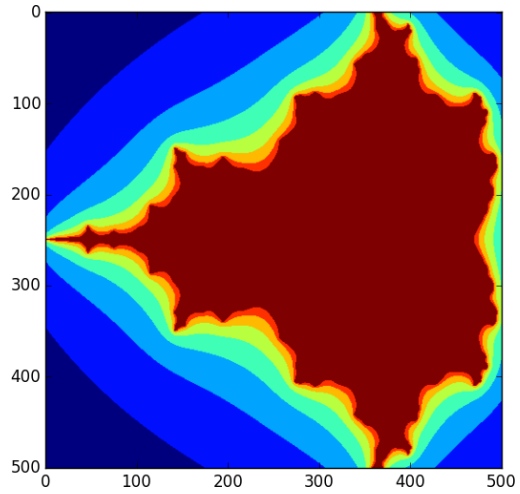

```
# Question 3
#===========
LX = [-2+2.5*x/100 for x in range(101)]
LY = [-1.1+2.2*x/100 for x in range(101)]
XY = [[[x,y] for x in LX] for y in LY]

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()
```

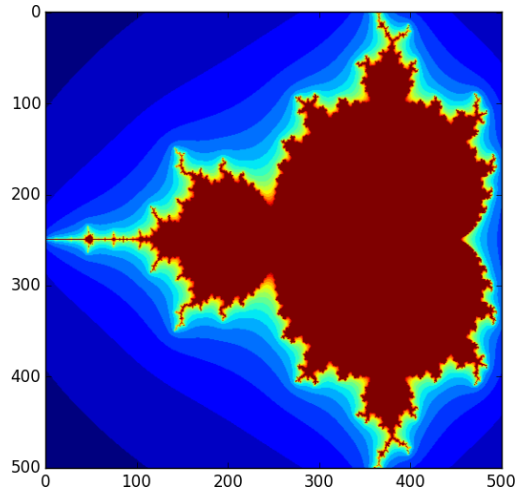
```
# Bilan
#=======
def affichage(m,M,res):
    m = m
    M = M
    LX = [-2+2.5*x/res for x in range(res+1)]
    LY = [-1.1+2.2*x/res for x in range(res+1)]
    XY = [[[x,y] for x in LX] for y in LY]
    for i in range(len(LX)):
        for j in range(len(LY)):
            XY[i][j]=fonction_f(
   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

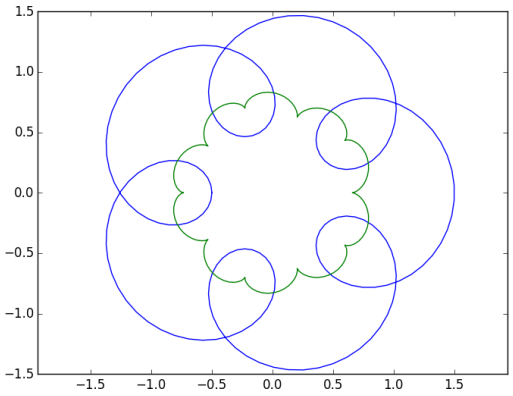
```
b,w = 0.5,6
# Question 1
# ==========
import numpy as np
def fonc_p(t):
    return [np.cos(t)+b*np.cos(w*t),np.sin(t)
                        +b*np.sin(w*t)]
```

```
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]
```

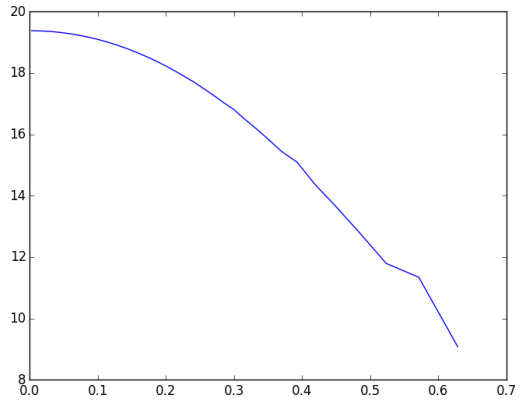
```
# Question 5
# ==========
les_xc = []
```

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

4

```
# 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 $\delta t$.*

# 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
```

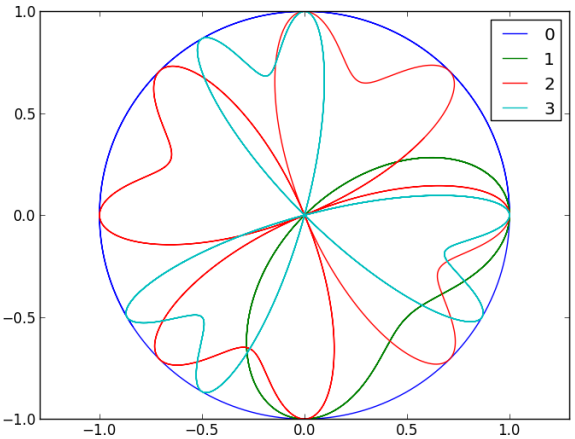

```
def mul_poly(P1,P2):
    """
    Calcule la multiplication de deux polynomes.
    Entrée :
     * P1, P2(lst) : liste des coefficients du
       polynome [a0,a1,...,an]
    Sortie :
     * P(lst) : produits des polynomes
    """
    P=[0]*(degre_poly(P1)+degre_poly(P2)+1)
    for i in range(len(P1)):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

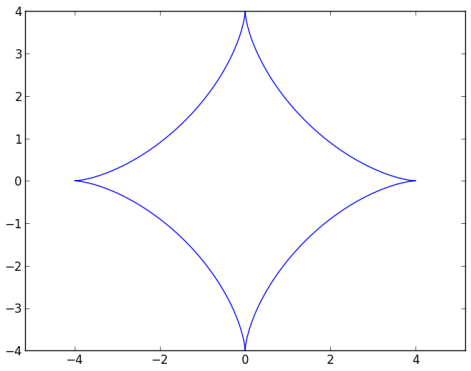
## Exercice 3 – Corrigé

```
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()
```

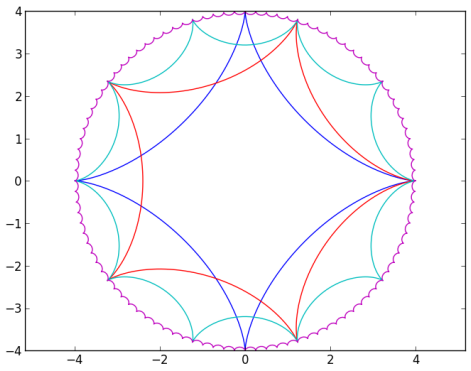



## Exercice 7 – Corrigé

```
# 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()
```



Oral

## Exercice 11 – Corrigé

```
# EXERCICE 10
# Question 1
# ==========
import matplotlib.pyplot as plt
import math

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

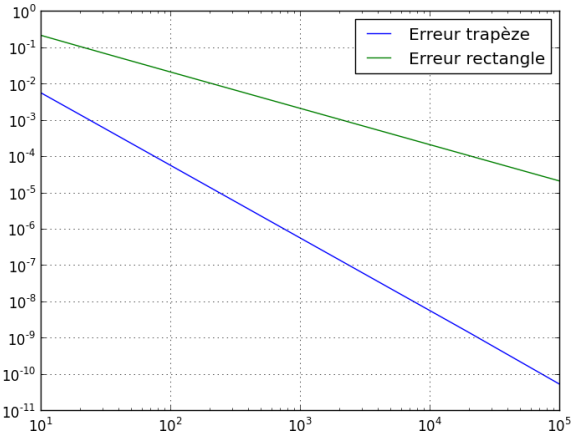
```
# Question 2
# ==========
def fonc(x):
    return math.log(x)
```

```
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 :

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

$$\Leftrightarrow \begin{cases} y_1(k+1) = hy_2(k) + y_1(k) \\ y_2(k+1) = h\left(\sin(nk) + y_1(k) - 10y_2(k)\right) + y_2(k) \end{cases}$$

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()
```