Lycée Faidherbe, 2020-2021

T.P. d'informatique

MPSI & PCSI

Version du 4 juin 2021

Table des matières

TP I

Introduction

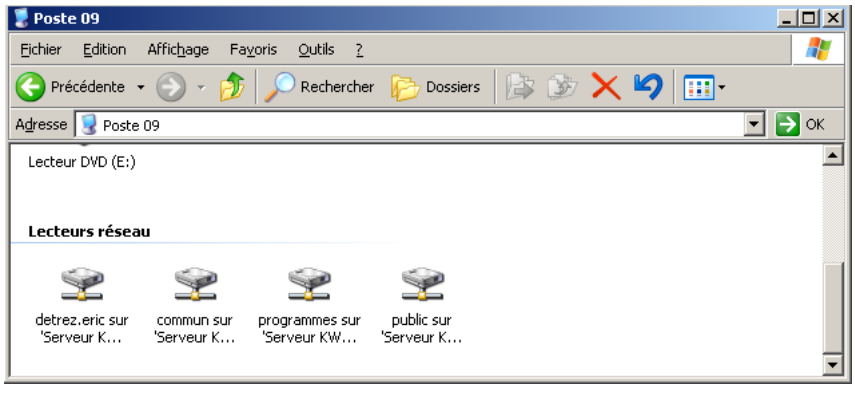
Résumé

Dans ce T.P. nous allons prendre contact avec l'environnement de travail utilisé durant l'année et expérimenter quelques ressources de python. Une grande partie des exercices seront approfondis durant le déroulement de l'année.
Pendant tous les T.P. il est capital de ne jamais rester passif : essayez, lisez les documentations, posez des questions à vos enseignants ou à ceux de vos camarades qui sont les plus rapides.

1 Écosystème

Nous utiliserons des outils dans un environnement qui sera, selon les séances, Windows ou Linux. Leurs fonctionnalités sont semblables mais il est utile de se familiariser avec chacun d'eux.
Pour utilisez les ordinateurs vous devez vous connecter avec votre identifiant (nom.prénom) et votre mot de passe. Le mot de passe initial est votre date de naissance et il **doit** être changé [1].

1.1 Répertoires et fichiers



Il y a 4 dossiers distants accessibles : Commun, Public, Programmes et votre dossier personnel, à votre nom.

- Sous Windows ils sont visibles comme des lecteurs réseaux, repérés par une lettre
- Sous Linux ce sont des répertoires.

Chacun a des modalités d'accès propres donc un usage spécifique.

1. Cela n'est possible que sur les ordinateurs sous Windows.

- Votre dossier personnel est le plus important. Vous seul y avez accès [2] en lecture (vous pouvez lire ce qui y est) et en écriture (vous pouvez y sauvegarder des fichiers). Les fichiers y seront conservés **toute l'année scolaire**. Il ne faut pas le confondre avec le dossier local de votre poste ; dans celui-ci aussi vous pouvez lire et écrire mais les données peuvent être effacées.
- Le dossier **public** est accessible à toute la classe, les étudiants y ont accès en lecture, les enseignants en lecture et en écriture. C'est le dossier dans lequel les enseignants peuvent transmettre des documents.
- Le dossier **commun** est accessible à toute la classe en lecture et en écriture. C'est le dossier dans lequel vous pouvez partager des documents avec les autres.
- Le dossier **programmes** contient des programmes (windows) qui peuvent être lancés. Il permet d'utiliser des programmes sur un poste windows sans qu'il soit installé sur le poste.

N.B. La taille totale des fichiers que vous pouvez sauvegarder est limitée (60 Mo) ; évitez les fichiers trop gros (films, musique, ...)

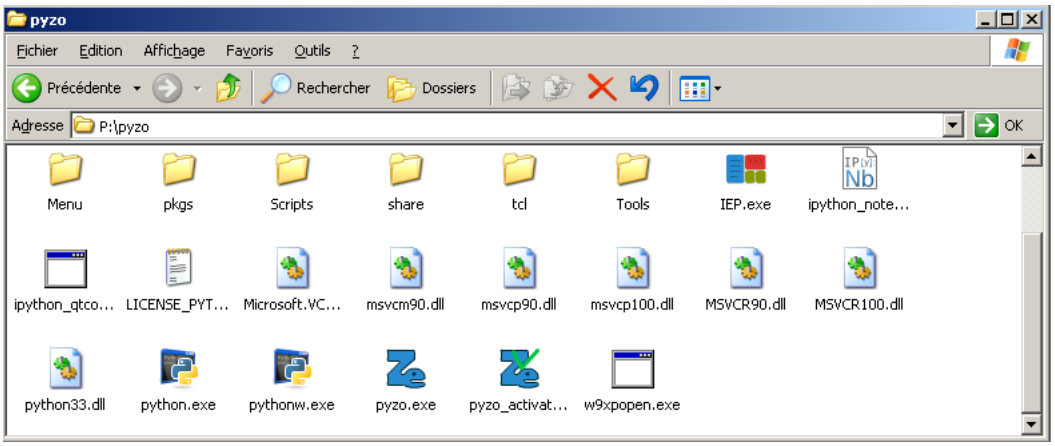
Créez un répertoire pour l'informatique dans votre dossier, entraînez-vous à y créer des sous-répertoires pour chaque TP.

2 Présentation de python

2.1 La boucle interactive

Le moyen le plus immédiat d'utiliser python est la boucle interactive lancée depuis un terminal.

- On peut trouver un terminal adapté, `winpython.exe` dans le dossier `Pyzo`

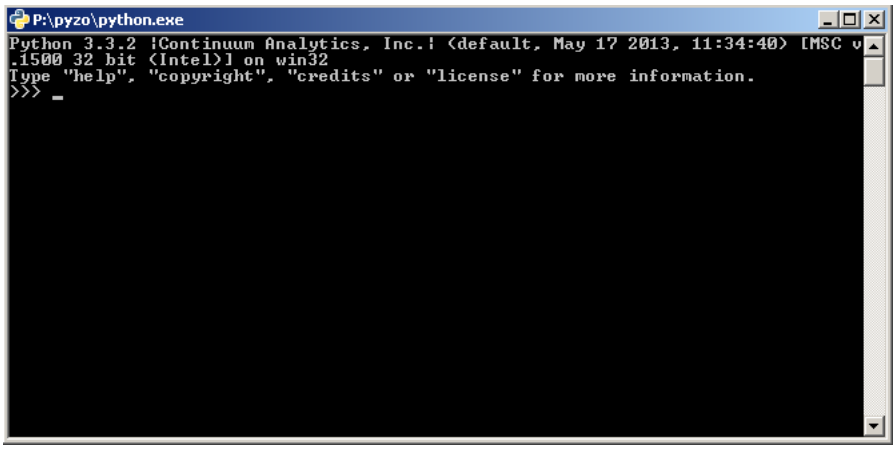


- Sur les postes Linux on ouvre le terminal par un clic droit sur un dossier "Ouvrir un terminal ici" puis on tape `python`. Python a eu une évolution importante en passant de la version 2 à la version 3 : une version 2.7 subsiste encore parfois mais nous utiliserons python3. Dans le cas où on aurait plusieurs versions de Python installées (2 et 3) on lance Python3 avec la commande `python3`.
  À notre niveau les différences ne sont pas nombreuses : la plus visible est que `print` est devenu une fonction et nécessite que ses arguments soient passés entre parenthèses.

Le programme python attend nos instructions et répond immédiatement.

On peut faire exécuter quelques instructions simples : `2 + 2`, ...

2. L'administrateur du réseau aussi.



On voit ici une particularité de python : c'est un langage interprété. Cela signifie qu'il ne fabrique pas, avec nos instructions, un programme autonome comme le font d'autres langages comme C ou Pascal. Il traduit nos commandes et les fait exécuter directement par l'ordinateur.

L'avantage est que l'on teste très facilement les programmes, l'inconvénient est qu'on a toujours besoin de python et qu'il lui faut tout traduire à chaque fois qu'on a besoin de notre programme.

On peut néanmoins lancer une suite d'instructions en python (un script) directement : on l'exécute par la commande `python "nom_du_fichier_avec_chemin"`.

2.2 Limitations

Cette boucle interactive est un peu primitive,

- il est difficile de gérer des programmes de plus d'une ligne (nous écrirons des programmes d'une dizaine de ligne),
- en particulier il sera difficile de corriger une erreur,
- on ne garde pas de trace de ce qui a été écrit.

Il existe une boucle interactive améliorée, souvent installée par défaut, `iPython`. Il y a une version autonome **iPython qtConsole** qui se lance dans une fenêtre, il existe aussi une version **notebook** qui se lance dans un navigateur. Vous trouverez sur internet des cours qui utilisent cette version.

Cependant la méthode la plus générale de programmation consiste à écrire le code dans un éditeur puis à le faire compiler ou interpréter par le langage. Dans le cas de l'interprétation d'un langage interactif l'interprétation peut se faire de manière accumulative : chaque nouveau code est ajouté et on enrichit petit à petit les outils qui sont à notre disposition.

Cette dernière méthode est possible dans Python.

On travaillera donc sous la forme d'un va-et-vient entre les instruction dans l'éditeur et leurs interprétations dans la console.

Il existe de nombreux logiciels qui intègrent un éditeur et une console Python, on parle d'**environnements de développement intégrés** ou IDE en anglais, pour Integrated Development Environment.

On trouve des IDE spécifiques à Python.

- **Pyzo** est utilisé par les oraux de Centrale, c'est celui qui a été choisi dans ce lycée.
- **Spyder** est assez semblable à Pyzo, certains le préfèrent.
- **idle** est souvent installé par défaut, il est utilisé à l'oral de l'ENSAM.
- **Thonny** est un outil à l'aspect épuré mais qui permet une installation facile des modules. De plus son mode de suivi de l'exécution (debug) est très instructif.

Les IDE généralistes (netBeans, Eclipse, Sublime Text, Atom, Visual Studio) peuvent être utilisés avec Python.

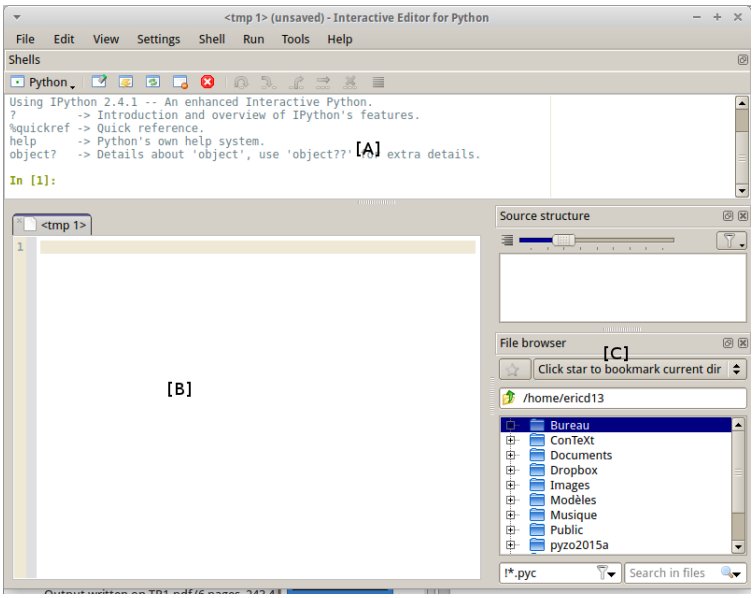
Les éditeurs de texte ont souvent, nativement ou à l'aide de greffons, la possibilité de travailler directement avec Python : emacs, gedit, notepad++ (windows), textmate (mac) ...

3 Pyzo

3.1 Présentation

La fenêtre initiale de Pyzo contient plusieurs parties :

- la console, [A], c'est l'interpréteur comme vu ci-dessus
- l'éditeur, [B], qui permet d'écrire les programmes et de les modifier
- des fenêtres d'utilitaires, [C], qui nous n'utiliserons pas pour l'instant.



On peut déplacer les composantes, sauf celle de l'éditeur, et fermer les outils qui ne sont pas utiles ; on pourra les retrouver en les ouvrant dans le menu **Outils**
Il est indispensable d'employer l'éditeur dès que l'on veut rentrer plusieurs lignes afin d'obtenir un résultat.
Il est par ailleurs recommandé que le fichier écrit dans l'éditeur soit complet :

1. on commence par les importations ; il n'est pas gênant qu'elles soit répétées, python ne les effectue qu'une fois
2. on poursuit par les différentes fonctions
3. on finit par les lignes principales qui appliquent ces fonctions.

On pourra utiliser un fichier-squelette à garder dans son répertoire.
Exemple (ce qui suit le caractère # n'est pas lu par Python).

```
# Importations


# Fonctions


# Programme principal
```

Par ailleurs il vaut mieux faire un fichier par exercice et il est **indispensable** de les sauvegarder régulièrement.

3.2 Le premier programme

Dans la fenêtre de l'éditeur effacer les lignes déjà écrites et écrire le programme le plus employé pour un premier exemple :

```
print("Hello World")
```

- On commence par sauvegarder le fichier en choisissant un emplacement que l'on peut retrouver et qui garde les données, le plus simple est le répertoire personnel du serveur du lycée que l'on peut lire partout dans le lycée.
- Il faut maintenant faire exécuter le fichier dans la console python
  - soit par un item *Exécuter le fichier* du menu *Exécuter*
  - soir par un raccourci ([F5] ou [Ctrl]+[Entrée])
- Tout se passe presque comme si on avait écrit les instructions dans la console.

Rappel : il est indispensable de sauvegarder régulièrement son travail. Pour cela il faut utiliser le raccourci-clavier [Ctrl] + S : on appuie sur la touche [Ctrl] puis, tout en maintenant l'appui sur [Ctrl], on appuie sur la touche S. On peut alors relâcher les deux touches.

3.3 Fonction `print`

En fait le comportement d'un programme écrit dans la console ou dans l'éditeur n'est pas exactement le même.
Lorsque l'on écrit une expression dans la console la valeur du résultat est affichée ; quand on le fait depuis un script le résultat est calculé mais n'est pas affiché.

```
>>> a = 1
>>> a
1
```

```
a = 1
a
----------------------------
>>>  (executing line ...)
```

Pour faire afficher le résultat il faudra utiliser la fonction `print`
Il y a plusieurs méthodes pour afficher le résultat dans un texte :

- On peut donner plusieurs arguments à `print`, séparés par des virgules. Python insère lui-même des espaces.

```
print("La somme de",a,"et",b,"vaut",a+b)
```

- On peut réserver des emplacements

```
print("La somme de {} et {} vaut {}".format(a,b,a+b))
```

Les deux affichent : `La somme de 3 et 4 vaut 7`

4 Activités

La plupart de ces activités peuvent être écrites dans la console. :

Entiers

Exercice I.1
Vérifier par quelques calculs que les opérations d'addition, `+`*, de multiplication,* `*` *ou d'exponentiation* `**` *fonctionnent comme prévu.*
Remarquer qu'il n'y a pas de limite à la taille des entiers, par exemple `2**100`*.*

Par contre la division avec `/` donne un résultat avec virgule, même dans le cas d'un entier divisé par un diviseur de cet entier (dans Python3). L'opération qui correspond à la division euclidienne est `//`, le reste est `n % p`.

Exercice I.2
Tester ces opérations ; par exemple `7 // 3` *et* `7 % 3`.

Réels

Dans python les réels sont appelés `float`. Il faut se souvenir que les `float` ne sont que des approximations décimales (en fait binaires) des réels souhaités.

Exercice I.3
Tester
- `0.1 + 0.2`
- `10 * 0.1`
- `0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1`
- `1.21**4.32` *(exponentiation)*
- `(5.0**0.5)**2.0` *cela devrait calculer* $\sqrt{5}^2$.

Dans python les fonctions et constantes mathématiques ne sont pas présentes par défaut. Il faut les importer depuis un module `math` (ou `numpy`).
- soit en chargeant le module par `import math` ; on appelle alors les fonctions par `math.cos`. Le logarithme népérien (ln) est `math.log`.
- soit en chargeant toutes les fonctions du module par[3] `from math import *` ; on appelle alors les fonctions par `exp`.
- soit en chargeant les fonctions utilisées une par une par `from math import tan` ; ici aussi, on peut alors appeler chaque fonction directement.

Exercice I.4
Calculer $\sin(\ln(2))$

Booléens

Les booléens sont des objets qui peuvent prendre la valeur `False` (faux) ou `True` (vrai).
Les opérations associées sont `and` (et), `or` (ou), `not` (non).

Exercice I.5
Après avoir donné la valeur 4 à la variable x tester `1 <= x and x <=5`.
Quelles sont les valeurs de x qui donnent le résultat `True` *?*

Remarque : Python permet d'écrire l'expression ci-dessus sous la forme `1 <= x <= 5`.

Exercice I.6
Il y a une priorité dans l'ordre des calculs :
évaluer `True or False and False`.
On doit parenthéser les `or` *si on veut qu'ils soient évalués en premier.*

Complexes

Python sait manipuler les complexes, sous la forme `x + yj` avec x et y réels ;
$1 + i$ se note `1.0 + 1.0j` (ou `1 + 1j`).
Quand on voudra définir un complexe avec des composantes calculées, il vaut mieux le construire à l'aide de la fonction `complex` :

```
>>> complex(1 + 2, 7/3
>>> (3+2.3333333333333335j)
```

Exercice I.7
Tester quelques calculs, par exemple $\frac{1+2i}{3+i}$

Les opérations non élémentaires sur les complexes sont dans un module `cmath` semblable à `math` dont nous ne nous servirons que peu.

Exercice I.8
Calculer $e^{i\pi}$

3. * est une abréviation pour dire tout

5 Compléments

5.1 Entrées-sorties

Lorsqu'on écrit un programme on peut souhaiter qu'il réponde à des entrées qui ne sont pas toujours les mêmes. Le programme principal peut donc interagir avec l'utilisateur en lui demandant d'entrer des paramètres. La fonction `input` est utile dans ce cas.
L'instruction `arg = input(ch)`

- affiche la chaîne de caractères `ch` (en général elle demande d'entrer la valeur de quelque chose),
- attend que l'on saisisse une chaîne de caractère au clavier suivie de l'appui de la touche **return**,
- puis affecte cette chaîne à la variable `arg`.

Le résultat est une chaîne de caractères, si on attend un nombre pour l'utiliser dans une expression, il faut convertir la chaîne à l'aide des fonctions `int` ou `float`.

Exercice I.9
Écrire les instructions qui permettent le dialogue suivant

```
Quel est ton nom ?
Taupin
Bonjour Taupin
```

Exercice I.10
Écrire les instructions qui permettent le dialogue suivant

```
Entrer le premier nombre :
3.54
Entrer le second nombre (non nul) :
2.27
3.54 + 2.27 = 5.8100000000000005
3.54 - 2.27 = 1.27
3.54 * 2.27 = 8.0358
3.54 / 2.27 = 1.5594713656387664
```

5.2 Fractions

Il existe un module, **`fractions`**, qui permet de gérer le cauchemar des collégiens : les fractions. Il ne contient qu'une fonction, `Fraction`, qui permet de définir une variable sous forme de fraction.

```
from fractions import Fraction
a = Fraction(2,4)
print(a)
----------------------
Fraction(1, 2)
```

On remarquera que la fraction est simplifiée.
On peut effectuer des opérations.

```
b = Fraction(1,6)
print(a+b)
----------------------
Fraction(2,3)
```

Nous verrons que le codage des réels les transforme en décimaux. On peut voir les réels sous leur forme de fraction.

```
>>> Fraction(math.sqrt(2))
Fraction(6369051672525773, 4503599627370496)
```

On peut même trouver une approximation avec un petit dénominateur.

```
>>> p = Fraction(math.pi)
>>> p
Fraction(884279719003555, 281474976710656)
>>> p.limit_denominator(50)
Fraction(22, 7)
```

La méthode `limit_denominator(n)` transforme la fraction par la meilleure approximation avec un dénominateur majoré par n.

Exercice I.11
Expérimentez

TP II

Variables

Résumé *Dans ce T.P. nous allons utiliser l'affectation et les bénéfices de l'usage des variables.*

Quelques rappels et conseils.

- Donnez à vos variables des noms parlants qui permettent de comprendre leur rôle. Ces noms ne doivent contenir que des lettres et des chiffres (pas de ponctuation, pas d'espace). Le caractère `_` (underscore) est aussi autorisé.
- Il pourra parfois être utile de stocker aussi des résultats de calcul intermédiaires en créant d'autres variables.
- La multiplication implicite (par exemple ab pour dire $a \times b$) n'est pas comprise par Python, donc utilisez forcément le symbole `*` pour les multiplications.
- La racine carrée peut s'écrire `**0.5`
- Les fonctions et constantes mathématiquesne sont pas disponibles par défaut. Ajouter la ligne `import math` au début de votre code. Vous disposerez alors de `math.pi` pour la valeur numérique de π, `math.sin` pour la fonction sinus ....
- Dans tous les exercices comportant des valeurs numériques, ne les utilisez jamais directement dans les calculs. Cela rend la relecture du code pénible (et donc les erreurs moins faciles à détecter) et si une valeur apparaît plusieurs fois vous allez la retaper inutilement. Stockez les valeurs numériques dans des variables avec la syntaxe `nom_variable = valeur`.
- L'ordinateur ne gère pas les unités, donc rentrez les valeurs numériques directement dans les bonnes unités, celles fournies conviennent le plus souvent.
- les puissances de 10 ont une écriture spécifique : 4.21×10^{-11} s'écrit `4.21e-11` (il faut que l'exposant soit entier et que les valeurs soient explicitement numériques.

1 Opérations simples

Exercice II.1
Voici un petit programme :

```
x = 10
y = 15
z = x + y
x = y
y = z
print(x + y + z)
```

Que va-t-il donner comme résultat ? Vérifier en l'exécutant depuis l'éditeur.

Exercice II.2
Que voudrait-on faire avec les lignes suivantes ?

```
x = 10
y = 15
y = x
x = y
```

Obtient-on le résultat souhaité ? Proposer une manière d'échanger les valeurs de deux variables.

2 Calculs physiques

2.1 Deuxième équivalence

On dose l'acide oxalique (un diacide) dont la concentration C est inconnue avec un volume initial $V_0 = 20 \times 10^{-3}\,\mathrm{L}$) par de la soude de concentration $C_b = 5 \times 10^{-2}\,\mathrm{mol\,L^{-1}}$. La première équivalence est peu visible, donc on ne peut exploiter que la seconde, pour laquelle :

$$2CV_0 = C_b V_{\text{éq2}} \quad \text{avec} \quad V_{\text{éq2}} = 12 \times 10^{-3}\,\mathrm{L}$$

Exercice II.3
Calculez numériquement C.

2.2 Pulsation plasma

À quelques dizaines de kilomètres d'altitude se trouve une couche d'atmosphère appelée *ionosphère*. Celle-ci est dans un état de la matière appelée *plasma* et ne laisse passer les ondes télécom que si leur fréquence est assez élevée. On donne :

$$\begin{cases} \nu = \dfrac{\omega}{2\pi} & \text{fréquence} \\ k^2 = \dfrac{\omega^2 - \omega_p^2}{c^2} & \text{nombre d'onde} \\ \omega_p = \sqrt{\dfrac{ne^2}{\varepsilon_0 m_e}} & \text{pulsation plasma} \end{cases} \quad \text{avec} \quad \begin{cases} e = 1.60 \times 10^{-19}\,\mathrm{C} \\ \varepsilon_0 = 8.85 \times 10^{-12}\,\mathrm{F\,m^{-1}} \\ m_e = 9.11 \times 10^{-31}\,\mathrm{kg} \\ n = 1.00 \times 10^{11}\,\mathrm{m^{-3}} \\ c = 3.00 \times 10^{8}\,\mathrm{m\,s^{-1}} \end{cases}$$

Exercice II.4
Calculez la fréquence minimale avec laquelle communiquer depuis la surface avec un satellite. Pour cela, il faut que le nombre d'onde soit réel.

2.3 Méthode de Bessel

Dans un home-cinéma, on veut projeter l'image d'un objet (le film) sur un écran (au mur) par une lentille. La distance objet-écran est imposée par la disposition de la salle et vaut $D = 5\,\mathrm{m}$. La lentille de projection a pour distance focale $f' = 0.4\,\mathrm{m}$.
Il existe deux positions de la lentille entre l'objet et l'écran telles que l'image est nette. Le grandissement (rapport entre la taille de l'objet et celle de l'image) vaut :

$$\gamma_1 = \frac{D + \sqrt{D^2 - 4Df'}}{-D + \sqrt{D^2 - 4Df'}} \quad \text{et} \quad \gamma_2 = \frac{D - \sqrt{D^2 - 4Df'}}{-D - \sqrt{D^2 - 4Df'}}$$

Exercice II.5
Calculez les deux grandissements. Vérification : leur produit doit valoir 1.

2.4 Résonance en transmission

Un électron d'énergie E arrive sur une zone de largeur l où son énergie potentielle vaut $V_0 < E$. D'après les lois de la physique quantique, la probabilité qu'il reparte en arrière (réflexion) est : [1]

$$R = \frac{(k_2^2 - k_1^2)^2 \sin(k_2 l)^2}{4k_1^2 k_2^2 + (k_1^2 - k_2^2)^2 \sin(k_2 l)^2} \quad \text{où} \quad \begin{cases} k_1 = \sqrt{\dfrac{2mE}{\hbar^2}} \\ k_2 = \sqrt{\dfrac{2m(E-V_0)}{\hbar^2}} \end{cases} \quad \text{avec} \quad \begin{cases} m = 9.11 \times 10^{-31}\,\text{kg} \\ E = 5.00\,\text{eV} \\ V_0 = 3.00\,\text{eV} \\ l = 5.00 \times 10^{-9}\,\text{m} \\ \hbar = 1.05 \times 10^{-34}\,\text{J s} \end{cases}$$

Dans k_1 et k_2, il faut exprimer les énergies E et V_0 en joules. Le facteur de conversion des électron-volt aux joules est $1\,\text{eV} = 1.60 \times 10^{-19}\,\text{J}$.

Exercice II.6
Calculez numériquement la probabilité R.

2.5 Angle de perte d'une bobine

Une bobine est un composant électronique caractérisé par deux coefficients réels positifs, son inductance L et sa résistance R. On peut alors définir son impédance, paramètre complexe défini par $Z = R + iL\omega$, avec ω la pulsation du signal électrique utilisé dans le montage.
On définit alors son *angle de perte* δ par :

$$\tan(\delta) = \frac{\text{Re}(Z)}{\text{Im}(Z)} \quad \text{avec} \quad \begin{cases} L = 0.1\,\text{H} \\ R = 10\,\Omega \\ \nu = 50\,\text{Hz} \\ \omega = 2\pi\nu \\ \delta \in \left[0, \dfrac{\pi}{2}\right[\end{cases}$$

La partie réelle et la partie imaginaire d'un complexe u s'obtiennent respectivement par `u.real` et `u.imag`.

Exercice II.7
Calculez l'angle de pertes en degrés.

3 Calculs mathématiques

Exercice II.8
Écrire un programme qui écrit une durée exprimée en secondes dans une variable `t` *sous la forme jours, heures, minutes, secondes.*
Par exemple, pour `t = 325415` *le programme devra afficher*
`325415 secondes correspondent à 3 jours 18h 23mn 35s`
Rappels : `//` *pour la division entière,* `%` *pour le reste.*

Exercice II.9
Le polynôme $aX^2 + bX + c$ est défini par trois variables réelles (de type `float`*)* `a`, `b` *et* `c`.
Écrire un programme qui affiche les deux racines du polynôme.
Que se passe-t-il lorsque le discriminant est négatif ?

1. Ce n'est pas demandé, mais le tracé de cette fonction $R(l)$ montre qu'il s'agit d'une fonction présentant des minima très piqués. Cela signifie que la probabilité complémentaire (de traverser la zone) $T = 1 - R$ présente un phénomène de *résonance* (passage par un maximum) pour certaines valeurs de l.

3.1 Polynômes de degré 3

Le but de ces exercices est de déterminer les 3 racines (complexes) d'un polynôme de degré 3 : $P(X) = X^3 + aX^2 + +bX + c$. Même dans le cas d'un polynôme à coefficients réels et dont les 3 racines sont réelles ont doit calculer des valeurs intermédiaires complexes.
On peut définir un nombre complexe à partir de son module r et de son argument φ à l'aide de la fonction `cmath.rect(r, phi)` du module `cmath`.

```
import cmath

>>> cmath.rect(2, cmath.pi/3)
(1.0000000000000002+1.7320508075688772j)
```

Exercice II.10
Étant donné un nombre (éventuellement complexe) dans une variable u, écrire les instructions qui permettent de calculer ses 3 racines troisièmes dans $\mathbb{C}$ *dans des variables* `a1`, `a2` *et* `a3`.

Dans le cas du polynôme $P(X) = X^3 + pX + q$ on démontre (exercice de mathématique) que si on pose $X = a + b$ en imposant $ab = -\frac{p}{3}$ alors $a^3 + b^3 = -q$. Comme on a aussi $a^3b^3 = -\frac{p^3}{27}$ on voit que a^3 et b^3 sont les racines de $Q(X) = X^2 + qX - \dfrac{p^3}{27}$.
On obtient alors 2 racines r_1 et r_2. On démontre aussi que si

- on choisit une des racines, r_1
- on attribue successivement à a les 3 racines de r_1
- on détermine b par $ab = -\frac{p}{3}$,

on obtient alors les trois racines de P en calculant $a + b$.

Exercice II.11
Écrire les instructions qui permettent de calculer les 3 racines, `t1`, `t2` *et* `t3` *du polynôme* X^3+pX+q *à partir des valeurs des coefficients donnés dans deux variables* `p` *et* `q`.
On supposera p non nul

Exercice II.12
Dans le cas général, $P(X) = X^3 + \alpha X^2 + \beta X + \gamma$, *montrer que u est racine de P si et seulement si* $u + \frac{\alpha}{3}$ *est racine d'un polynôme de la forme* $P_1(X) = X^3 + pX + q$ *dont on déterminera les coefficients p et q en fonction de* α, β *et* γ.
En déduire le calcul des 3 racines du polynôme P à partir des variables `alpha`, `beta` *et* `gamma` *dans trois variables* `t1`, `t2` *et* `t3`
On pourra supposer qu'on a $p \neq 0$ *ou traiter ce cas en utilisant une instruction conditionnelle.*

4 Solutions

Solution de l'exercice II.3 -

```
V0 = 20e-3
Veq2 = 12e-3
Cb = 5e-2
C = Cb*Veq2/(2*V0)
print(C)
```

On trouve $C = 1.5 \times 10^{-2}\,\mathrm{mol\,L^{-1}}$.

Solution de l'exercice II.4 - On veut $k^2 \geqslant 0$ et donc $\omega_{\min} = \omega_p$. D'où

```
e = 1.6e-19
eps0 = 8.85e-12
me = 9.11e-31
n = 1e11
c = 3e8

wp2 = n*e**2/(eps0*me)
fmin = wp2**0.5/(2*math.pi)
print(fmin)
```

On trouve $\nu = 2.83 \times 10^{6}\,\mathrm{Hz}$.

Solution de l'exercice II.5 - Pour minimiser les répétitions du même calcul (la racine du discriminant) on crée une variable intermédiaire.

```
D = 5
fp = 0.4
sq_delta = (D**2-4*D*fp)**0.5

gamma1 = (D+sq_delta)/(-D+sq_delta)
gamma2 = (D-sq_delta)/(-D-sq_delta)
print(gamma1)
print(gamma2)
print(gamma1*gamma2)
```

On trouve $\gamma_1 = -10.4$ et $\gamma_2 = -0.09$.

Solution de l'exercice II.6 -

```
m = 9.11e-31
hbar = 1.05e-34
l = 5e-9
conv_eV = 1.6e-19
E = 5*conv_eV
V0 = 3*conv_eV

k12 = 2*m*E/hbar**2
k22 = 2*m*(E-V0)/hbar**2
a = (k22-k12)**2*math.sin(k22**0.5*l)**2
R = a/(4*k12*k22+a)
print(R)
```

On trouve $R = 0,176$.

Solution de l'exercice II.7 -

```
L = 0.1
R = 10
nu = 50

omega = 2*math.pi*nu
Z = R+1j*L*omega
tand = Z.real/Z.imag
delta_rad = math.atan(tand)
delta_deg = delta_rad*180/math.pi
print(delta_deg)
```

On trouve $\delta = 17.7°$. On peut aussi remarquer que c'est le complémentaire de $\arg(Z)$.

Solution de l'exercice II.10 -

```
a1 = u**(1/3)
w = cmath.rect(1,2*cmath.pi/3)
a2 = a1*w
a3 = a2*w # ou a1/w
```

Solution de l'exercice II.11 -

```
delta  = q**2 + 4*p**3/27
u = (-q + delta**(1/2))/2
a1 = u**(1/3)
w = cmath.rect(1,2*cmath.pi/3)
a2 = a1*w
a3 = a2*w
b1 = -p/3/a1
b2 = -p/3/a2
b3 = -p/3/a3
t1 = a1 + b1
t2 = a2 + b2
t3 = a3 + b3
```

Solution de l'exercice II.12 -
$\left(u+\frac{\alpha}{3}\right)^3 = u^3+\alpha.u^2+\frac{\alpha^2}{3}u+\frac{\alpha^3}{27}$ or $u^3+\alpha.u^2 = -\beta.u-\gamma$ car u est racine.
Ainsi $\left(u+\frac{\alpha}{3}\right)^3 = \left(\frac{\alpha^2}{3}-\beta\right)u+\frac{\alpha^3}{27}-\gamma = \left(\frac{\alpha^2}{3}-\beta\right)\left(u+\frac{\alpha}{3}\right)+\frac{\alpha^3}{27}-c-\frac{\alpha^3}{9}+\frac{\alpha\beta}{3}$.
$p = \beta-\frac{\alpha^2}{3}$ et $q = \gamma-\frac{\alpha\beta}{3}+\frac{2\alpha^3}{27}$.

```
import cmath

alpha = -6
beta = 11
gamma = -6

p = beta - alpha**2/3
q = gamma - alpha*beta/3 + 2*alpha**3/27
d = -alpha/3
if p == 0:
    u = -q
    t1 = u**(1/3) + d
    w = cmath.rect(1,2*cmath.pi/3)
    t2 = t1*w + d
    t3 = t2*w  + d
else:
    delta  = q**2 + 4*p**3/27
    u = (-q + delta**(1/2))/2
    a1 = u**(1/3)
    w = cmath.rect(1, 2*cmath.pi/3)
    a2 = a1*w
    a3 = a2*w
    b1 = -p/3/a1
    b2 = -p/3/a2
    b3 = -p/3/a3
    t1 = a1 + b1 + d
    t2 = a2 + b2 + d
    t3 = a3 + b3 + d
print(t1, t2, t3)
```

TP III

Fonctions

Résumé

Dans ce T.P. nous allons utiliser des fonctions utiles de python, définir les premières fonctions et utiliser les instructions de branchement.
Dans un premier temps on donnera un cadre de présentation du code écrit dans l'éditeur qu'il conviendra de respecter à chaque T.P.

1 Mise en page dans l'éditeur

Dans les T.P. nous importerons des fonctions définis dans des modules, nous définirons nos propres fonctions et nous utiliserons ces fonctions pour obtenir des résultats.
Il est utile de structurer le code écrit. Pour cela chaque T.P. on commencera donc par indiquer, par un commentaire, les parties selon la forme ci-dessous.

```
## Importations



## Fonctions

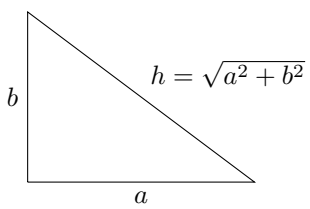


## Principal
```

Importations est l'endroit où on importe les modules, par exemple `from math import *`
C'est aussi ici que l'on pourra charger des fichiers et définir des constantes
Fonctions est l'endroit où on écrit les fonctions.
Principal est l'endroit où on écrit les scripts qui utilisent les fonctions.

2 Vers des fonctions utiles

On cherche à écrire une fonction qui calcule l'hypoténuse d'un triangle rectangle à partir des longueurs des côtés.



2.1 Un calcul vain

Jules Jefonce écrit la fonction suivante

```
def hypo1(a, b):
    """Entrées : deux nombres
       Sortie : l'hypothénuse du triangle rectangle de cotes a
            et b"""
    (a**2 + b**2)**0.5
```

Exercice III.1
Peut-on obtenir le résultat de `hypo1(4, 7)` *?*
On pourra essayer `print(hypo1(4, 7))` *dans la partie principale et* `hypo1(4, 7)` *dans la console.*

Jules se souvient que les résultats sont perdus si on ne les mémorisent pas dans une variable ; il écrit alors

```
def hypo2(a, b):
    """Entrées : deux nombres
       Sortie : l'hypothénuse du traingle rectangle de cotes a
            et b"""
    h = (a**2 + b**2)**0.5
```

Exercice III.2
La variable `h` *est-elle accessible après le calcul de* `hypo2(4, 7)` *?*
Vérifier dans le Workspace de Pyzo ou l'onglet des variables de Thonny.

2.2 Voir le résultat

Sandrine Lafutée, la voisine de Jules, lui dit, "C'est normal, tout ce qui est dans la fonction est oublié, il faut afficher le résultat depuis la fonction".
Cette entraide leur permet d'écrire

```
def hypo3(a, b):
    """Entrées : deux nombres
       Sortie : l'hypothénuse du traingle rectangle de cotes a
            et b"""
    h = (a**2 + b**2)**0.5
    print(h)
```

Ils essayent leur fonction en écrivant `hypo3(4, 7)` et ça marche.

Exercice III.3
On utilise la fonction dans la partie principale : `print(hypo3(4, 7))`.
Est-ce le résultat attendu ?

2.3 Utiliser le résultat

Ils passent outre le petit souci précédent et passent à la question suivante : il faut tester si 5, 12 et 13 sont les côtés d'un triangle rectangle. Ils écrivent donc le test dans la console et obtiennent [1]

```
>>> hypo3(5, 12) == 13
13
False
```

1. Python est laxiste ici, il ne devrait pas indiquer `False` mais renvoyer une erreur.

Exercice III.4
Écrire une fonction `hypo4` *qui peut être utilisée.*

Moralité : **si on demande de renvoyer un résultat, on le renvoie avec** `return`.

3 Fonctions

3.1 Fonctions simples

Exercice III.5 — Polynôme
Écrire une fonction `P` *telle que* `P(a)` *renvoie la valeur du polynôme* $X^3 - 7.2X + 1.4$ *en* a.

Exercice III.6 — Maximum
La fonction `max` *renvoie le maximum de 2 nombres.*
Écrire une fonction `max3` *renvoyant le plus grand parmi* 3 *nombres.*

Exercice III.7 — Devinette
Importer le module `math` *afin de pouvoir utiliser la constante* `pi` *et la fonction* `floor`.
Tester la fonction suivante pour `nombre = pi` *et* `n = 1`, `n = 2`, `n = 3`.

```
def devine(nombre, n):
    a = 10**(n-1)*nombre
    b = a - floor(a)
    return floor(10*b)
```

Que fait cette fonction ?

Une fonction déjà écrite peut bien entendu être utilisée dans une fonction.

Exercice III.8 — Taux d'accroissement
Écrire une fonction `acc_P(a, b)` *qui renvoie le taux d'accroissement de* P *entre* a *et* b *:* $\dfrac{P(b) - P(a)}{b - a}$.
P *est le polynôme défini à la question III.5.*

3.2 Résultats booléens

Lorsque l'on fait une comparaison, l'expression renvoie un booléen qui peut être retourné. Par exemple

```
def est_positif(x):
    return x > 0
```

Les fonctions de cette partie doivent être écrites sans utiliser l'instruction conditionnelle **`if`**.

Exercice III.9 — Parité
Écrire une fonction `est_pair` *telle que* `est_pair(n)` *renvoie* `True` *ou* `False` *selon que l'entier* n *est pair ou impair.*

Exercice III.10 — Intervalle
Écrire une fonction `entre` *telle que* `entre(a, b, x)` *renvoie* `True` *ou* `False` *selon que le réel* x *est appartient à l'intervalle* $[a; b]$ *ou non. La fonction* **ne doit pas** *utiliser l'instruction* `if`.
Remarque : PYTHON *permet d'écrire la fonction sans utiliser* `and`.

Une année est **bissextile**, c'est-à-dire qu'elle comporte 366 jours au lieu de 365 si elle est divisible par 4 sauf tous les centenaires (1700, 1800, 1900, ..). Cependant tous les multiples de 400 sont quand même bissextiles.

Exercice III.11 — Années bissextiles
Écrire une fonction `bissextile(n)` *qui renvoie* `True` *ou* `False` *selon que l'année n est bissextile ou non.*
On peut écrire une fonction sur 2 lignes : (`def` *et* `return`*).*

4 Paramètres

Un fonction peut ne pas avoir de variable : on peut écrire une fonction qui fait une suite constante d'instructions, on peut simuler un événement aléatoire, ...
Le module `random` contient une fonction `randint` telle que `random.randint(a,b)` renvoie un entier choisi au hasard entre a et b (bornes comprises).

Exercice III.12 — Tirage de dé
Écrire une fonction sans variable `tirageDe()` *qui simule un tirage de dé.*
Faire quelques appels de cette fonction.

Une caractéristique des langages modernes comme python est qu'ils admettent les fonctions comme des paramètres possibles.

Exercice III.13 — Taux d'accroissement général
On généralise l'exercice III.8 : écrire une fonction `acc(f, a, b)` *qui renvoie le taux d'accroissement d'une fonction f passée en paramètre entre a et b :* $\dfrac{f(b)-f(a)}{b-a}$.

On sait que la dérivée d'une fonction f en x est la limite quand h tend vers 0 du taux d'accroissement $\dfrac{f(x+h)-f(x)}{h}$: on peut utiliser un taux d'accroissement avec une valeur h "petite" pour approcher la dérivée. Cependant, il plus efficace d'approcher la dérivée par un taux d'accroissement symétrique : $\dfrac{f(x+h)-f(x-h)}{2h}$.

Exercice III.14 — Dérivée
Écrire une fonction `derivee(f, x)` *qui approche $f'(x)$ par un taux d'accroissement symétrique ; on prendra $h = 10^{-5}$.*

On peut utiliser des paramètres qui ont une valeur par défaut mais que l'on peut modifier ; ils sont définis dans les paramètres avec une affectation : `def f(x, y, z = 3)`.

Exercice III.15 — Dérivée à pas modifiable
Modifier la fonction `derivee(f, x)` *pour que l'écart soit un paramètre optionnel de valeur 10^{-5}.*
Comparer la dérivée de $\sin(1)$ *avec* $\cos(1)$ *pour différentes valeurs de h.*

Hors programme : on peut abstraire le calcul ci-dessus en créant une fonction qui reçoit une fonction f est qui renvoie une approximation de sa dérivée *en tant que fonction*. Il suffit de créer la fonction dérivée dans le corps de la fonction et de la renvoyer.

Exercice III.16 — Fonction dérivée
Écrire une fonction `fn_derivee(f)` *qui renvoie f' avec l'approximation par un taux d'accroissement symétrique avec un pas de 10^{-5}.*

5 Solutions

Solution de l'exercice III.1 - Rien ne marche, on voit apparaître des `None`.

Solution de l'exercice III.2 - `h` a disparu

Solution de l'exercice III.3 - En plus du résultat, on voit apparaître un `None`

Solution de l'exercice III.4 -

```
def hypo4(a, b):
    """Entrées : deux nombres
       Sortie : l'hypothénuse du traingle rectangle de cotes a
           et b"""
    h = (a**2 + b**2)**0.5
    return h
```

Solution de l'exercice III.5 -

```
def P(x):
    return x**3 - 7.2*x + 1.4
```

Solution de l'exercice III.6 -

```
def max3(x, y, z):
    a = max(x, y)
    return max (a, z)
```

Solution de l'exercice III.7 - Elle calcule la n-ième décimale du nombre.

Solution de l'exercice III.8 -

```
def acc_P(a, b)):
    return (P(b) - P(a))/(b-a)
```

Solution de l'exercice III.9 -

```
def est_pair(n):
    return n%2 == 0
```

Solution de l'exercice III.10 -

```
def entre(a, b, x):
    return a <= x and x <= b
```

L'écriture suivante est possible

```
def entre(a, b, x):
    return a <= x <= b
```

Solution de l'exercice III.11 -

```
def bissextile(n):
    return (n%4 == 0 and n%100 != 0) or n%400 == 0
```

Solution de l'exercice III.12 -

```
from random import randint

def tirageDe():
    k = randint(1,6)
    return k
```

Solution de l'exercice III.13 -

```
def acc_P(a, b)):
    return (P(b) - P(a))/(b-a)
```

Solution de l'exercice III.14 -

```
def derivee(f, x):
    h = 1.0e-5
    acc = (f(x+h) - f(x-h))/(2*h)
    return acc
```

Solution de l'exercice III.15 -

```
def derivee(f, x, h = 1.0e-5):
    acc = (f(x+h) - f(x-h))/(2*h)
    return acc
```

```
import math as m

print(derivee(m.sin, 1, 1e-3) - m.cos(1)
print(derivee(m.sin, 1, 1e-4) - m.cos(1)
print(derivee(m.sin, 1, 1e-5) - m.cos(1)
print(derivee(m.sin, 1, 1e-6) - m.cos(1)
print(derivee(m.sin, 1, 1e-7) - m.cos(1)
print(derivee(m.sin, 1, 1e-8) - m.cos(1)
print(derivee(m.sin, 1, 1e-9) - m.cos(1)
print(derivee(m.sin, 1, 1e-10) - m.cos(1)
```

On voit que l'erreur augmente au-delà de 10^{-6}, les erreurs d'arrondi rendent la précision plus mauvaise.

Solution de l'exercice III.16 -

```
def fn_derivee(f):
    h = 1.0e-5
    def df(x):
        return (f(x+h) - f(x-h))/(2*h)
    return df
```

TP IV

Boucles 1 : for

Résumé *Dans ce T.P. nous allons mettre en œuvre la répétitions des instructions avec une boucle* `for`.

1 Répétitions simples

L'usage le plus simple d'une boucle `for` est de faire exécuter un certain nombre de fois les mêmes instructions. On emploie la structure `for i in range(n)` pour répéter n fois, la variable i n'est pas utilisée [1]

Exercice IV.1 — La punition
Écrire une fonction `ecrirePunition(texte,n)` qui reçoit deux paramètres
- `texte` est une chaîne de caractères
- `n` est un entier

et qui écrit n fois le texte à l'écran. Exceptionnellement, il n'y a pas de `return`.

Exercice IV.2 — Une suite
Écrire une fonction `heron(n, u0)` qui calcule (et renvoie) le terme u_n de la suite (u_p) définie par $u_0 =$ `u0` et $u_{p+1} = \frac{1}{2}\left(u_p + \frac{2}{u_p}\right)$ (suite de Héron).

Exercice IV.3 — Suite de Fibonacci : 1
On considère la suite (de Fibonacci) définie par $F_0 = 0$, $F_1 = 0$ et $F_{n+2} = F_{n+1} + F_n$ pour $n \geqslant 0$.
Écrire un programme `fibo(n)` qui renvoie F_n.
On pourra maintenir deux variables qui correspondent à 2 valeurs consécutives de la suite.

2 Utilisation de l'indice de boucle

Exercice IV.4 — Table de multiplication
Écrire une fonction `table(n` qui écrit la table de multiplication par n.

```
1 fois 7 donne 7
2 fois 7 donne 14
...
```

Exercice IV.5 — Suite de Fibonacci : 2
Affichez les valeurs de F_0 à F_{12}.

1. On pourrait écrire `for _ in range(n)`, le caractère "_" représente une variable sans nom.

Exercice IV.6 — Suite harmonique

Écrire une fonction `harmo(n)` qui renvoie $\sum_{k=1}^{n} \frac{1}{k}$ (pour $n \geqslant 1$).

Exercice IV.7 — Factorielle

Écrire une fonction `facto(n)` qui renvoie $n!$.

Utiliser cette fonction pour écrire une fonction `binomial(n,p)` qui renvoie $\binom{n}{p}$.

Exercice IV.8 — Coefficients binomiaux, bis

La fonction précédente manipule des entiers qui peuvent être beaucoup plus grands que le résultat. Si on n'a pris soin d'écrire des divisions entières (//) `binomial(1000, 1)` donne une erreur.

Écrire une fonction `binomial(n,p)` qui renvoie $\binom{n}{p}$ en utilisant la propriété $\binom{n}{k+1} = \frac{n-k}{k+1}\binom{n}{k}$.

Exercice IV.9 — Somme de puissances

Écrire une fonction `sommePuissances(n,p)` qui renvoie $\sum_{k=1}^{n} k^p$.

Calculer, pour quelques valeurs de n, `sommePuissances(n,3)-(sommePuissances(n,1))**2`.

Exercice IV.10 — Série

On veut étudier la suite défine par la somme $S_n = \sum_{k=0}^{n} \frac{2^k}{k!}$.

- Écrire une fonction `a(n)` qui calcule $a_n = \frac{2^n}{n!}$; en déduire une fonction `S(n)` qui calcule S_n. Combien d'opérations sont effectuées lors de l'appel de `S(n)` ?
- Proposer une fonction `S1(n)` qui calcule S_n en effectuant un nombre d'opération proportionnel à n (en fait proportionnel à $n+1$).

Exercice IV.11 — Constante d'Euler

Les deux suites $(u_n)_{n\geqslant 1}$ et $(v_n)_{n\geqslant 1}$ définies par $u_n = H_n - \ln(n+1)$ et $v_n = H_n - \ln(n)$ (voir refexo :harmo) sont adjacentes ; leur limite commune est la constante d'Euler, notée γ. On a donc $u_n \leqslant \gamma \leqslant v_n$ pour tout n.

On prouve, de plus qu'on a $v_n - u - n \leqslant \frac{1}{n}$.

Écrire une fonction `gamma(e)` qui calcule un encadrement de γ avec une précision e.

2.1 Suite de Muller (1993)

La suite de Muller *est définie par* $u_0 = \frac{5}{2}$, $u_1 = \frac{17}{5}$ *et* $u_{n+2} = 30 - \frac{129}{u_{n+1}} + \frac{100}{u_n.u_{n+1}}$ *pour* $n \geqslant 0$.

Exercice IV.12 — Calcul des termes

Écrire une fonction `muller(n)` qui calcule le terme u_n de cette suite.

Calculer les 50 premières valeurs de u_n ; quelle semble être la limite ?

On prouve, par récurrence, que $u_n = \frac{1+4^{n+1}}{1+4^n}$ *: la suite converge vers 4.*

On peut utiliser des fractions ; elles sont utilisables comme les nombres.

```
from fractions import Fraction
a = Fraction(2, 6)
print(a, a*2, float(a))
```

Exercice IV.13 — Usage des fractions

Écrire une fonction `muller_frac(n)` qui calcule le terme u_n de la suite de Muller en utilisant les fractions.

3 Exercices supplémentaires

Ces exercices sont destinés à ceux qui auraient achevé les 4 premiers T.P. et désireraient s'exercer sur des problèmes plus difficiles.

3.1 Polynômes de degré 3, reprise du TP02

Le but de ces exercices est de déterminer les 3 racines (complexes) d'un polynôme de degré 3 : $P(X) = X^3 + aX^2 + +bX + c$. Même dans le cas d'un polynôme à coefficients réels et dont les 3 racines sont réelles ont doit calculer des valeurs intermédiaires complexes.
On peut définir un nombre complexe à partir de son module r et de son argument φ à l'aide de la fonction `cmath.rect(r, phi)` du module `cmath`.

```
import cmath

>>> cmath.rect(2, cmath.pi/3)
(1.0000000000000002+1.7320508075688772j)
```

Il sera pratique de placer les racines dans une liste `r = [0]*3` dont les éléments seront accessibles par `r[0]`, `r[1]` et `r[2]`.

Exercice IV.14
Écrire une fonction `racine3(u)` *qui renvoie les 3 racines (complexes) de u.*
Si a_1 est la valeur calculée par `u**(1/3)`*, les deux autres racines sont $a_2 = a_1 e^{2i\pi/3}$ et $a_3 = a_1 e^{4i\pi/3}$*

Dans le cas du polynôme $P(X) = X^3 + pX + q$ on démontre (exercice de mathématique) que si on pose $X = a + b$ en imposant $ab = -\frac{p}{3}$ alors $a^3 + b^3 = -q$. Comme on a aussi $a^3 b^3 = -\frac{p^3}{27}$ on voit que a^3 et b^3 sont les racines de $Q(X) = X^2 + qX - \dfrac{p^3}{27}$.
On obtient alors 2 racines r_1 et r_2. On démontre aussi que si

- on choisit une des racines, r_1
- on attribue successivement à a les 3 racines de r_1 comme valeur,
- on détermine b par $ab = -\frac{p}{3}$,

on obtient alors les trois racines de P en calculant $a + b$.

Exercice IV.15
Écrire une fonction `solutions1(p, q)` *qui calcule les 3 racines du polynôme $X^3 + pX + q$. On supposera p non nul.*

Exercice IV.16
Dans le cas général, $P(X) = X^3 + aX^2 + bX + c$, montrer que u est racine de P si et seulement si $u + \frac{a}{3}$ est racine d'un polynôme de la forme $P_1(X) = X^3 + pX + q$ dont on déterminera les coefficients p et q en fonction de a, b et c.

Exercice IV.17
En déduire une fonction `solutions(a, b, c)` *qui calcule les 3 racines du polynôme $P = X^3 + aX^2 + bX + c$.*

3.2 6400 carrés dans un presque-carré

Ce problème est le défi Turing 8 : http ://apprendre-en-ligne.net/
Dans un rectangle de longueur 4 et de largeur 3, on peut dessiner 12 carrés de côté 1, 6 carrés de côté 2 et 2 carrés de côté 3. Au total, on peut dessiner 20 carrés. Nous dirons que c'est un rectangle-20. Les dimensions sont entières tant pour le rectangle que pour les carrés.

Exercice IV.18
Quelle est l'aire du rectangle-6400 dont la forme est la plus proche d'un carré ?

3.3 Longue suite de Syracuse

Ce problème est le défi Turing 14 : http ://apprendre-en-ligne.net/
Il est aussi le problème 14 du Project Euler : https ://projecteuler.net/problem=14

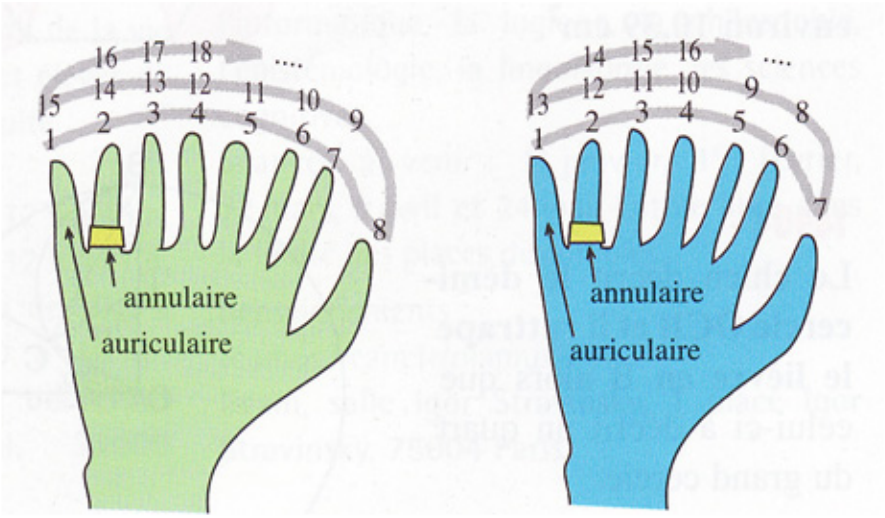
La suite de Syracuse est définie par $u_0 \in \mathbb{N}^*$ et $u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ 3u_n + 1 & \text{si } u_n \text{ est impair} \end{cases}$

On conjecture que la suite de Syracuse de n'importe quel entier strictement positif atteint 1.
Par exemple, à partir de 14, on construit la suite des nombres :
14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1.
C'est ce qu'on appelle la suite de Syracuse du nombre 14. Elle a ici une longueur de 18.

Exercice IV.19
Pour quel nombre de départ inférieur à 1 500 000 obtient-on la plus longue suite de Syracuse ? Il y a deux solutions, donner la plus petite.

3.4 Rencontre du quatrième type

Ce problème est le défi Turing 19 : http ://apprendre-en-ligne.net/
Des petits hommes verts rencontrent des petits hommes bleus. A leur grand étonnement, ils constatent que leurs mains ne comportent pas le même nombre de doigts : 7 pour les bleus et 8 pour les verts. Mais les savants des deux peuples remarquent que si l'on compte sur les doigts comme indiqué sur la figure, en faisant des allers-retours de l'auriculaire vers le pouce, puis du pouce vers l'auriculaire, certains nombres se comptent à la fois sur l'annulaire des mains bleues et sur celui des mains vertes (le 2 et le 14 par exemple). Ces nombres ont été qualifiés d'"annulaires" par les savants.



Exercice IV.20
Calculer la somme des nombres annulaires compris entre 1 et 2020.

3.5 Nombre divisible par morceaux

Ce problème est le défi Turing 26 : http ://apprendre-en-ligne.net/

Exercice IV.21
Trouver un nombre entier $c_1c_2c_3c_4c_5c_6c_7c_8c_9$ composé de tous les chiffres de 1 à 9, tel que $c_1c_2\cdots c_k$ soit divisible par k, pour tout $k \in \{1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
On pourra utiliser `permutations` *du module* `itertools`.

Plus simplement, pour 3 chiffres, 321 convient.
3 est divisible par 1 32 est divisible par 2 321 est divisible par 3.

4 Solutions

Solution de l'exercice IV.1 -

```
def ecrirePunition(texte,n):
      """Entrées : une chaîne de caractères et un entier
         Sortie  : la chaîne est écrite n fois"""
    for i in range(n):
        print(texte)
```

Solution de l'exercice IV.2 -

```
def heron(n,u0):
      """Entrées : un entier positif n et un réel u0
         Sortie  : le n-ième terme de la suite de Héron"""
    u = u0
    for i in range(n):
        u = (u + 2/u)/2
    return u
```

Solution de l'exercice IV.3 -

```
def fibo(n):
    """Entrée : un entier positif n
       Sortie : le n-ième nombre de Fibonacci"""
    F = 0
    F_suivant = 0
    for i in range(n): # On calcule les couples (F1, F2), (F2,
        F3), ..., (Fn,F(n+1))
        F_vieux = F
        F = F_suivant
        F_suivant = F + F_vieux
    return F
```

Il est à noter que l'on a calculé un terme de plus : on pourrait être tenté d'écrire

```
def fibo(n):
    """Entrée : un entier positif n
       Sortie : le n-ième nombre de Fibonacci"""
    F = 0
    F_suivant = 0
    for i in range(n-1): # On calcule les couples (F1, F2), (
       F2, F3), ..., (F(n-1), Fn)
        F_vieux = F
        F = F_suivant
        F_suivant = F + F_vieux
    return F_suivant
```

La valeur de `fibo(0)` serait juste mais uniquement car $F_0 = F_1$..

Solution de l'exercice IV.4 -

```
def table(n):
   for k in range(1, 11):
       print("{} fois {} donne {}.format(n, k, k*n))
```

Solution de l'exercice IV.5 -

```
for i in range(13):
    print("F{} vaut {}".format(i, fibo(i)))
```

Solution de l'exercice IV.6 -

```
def harmo(n) :
      """Entrée : un entier strictement positif
         Sortie : la valeur de Hn"""
    h=0
    for k in range(n):
        h = h + 1/(k+1)
    return k
```

Solution de l'exercice IV.7 - Il faut initialiser par 1 car on fait des produits.

```
def facto(n):
      """Entrée : un entier positif
         Sortie : la valeur de n!"""
    f = 1
    for k in range(n):
        f = f*(k+1)
    return f
```

```
def binomial(n,p):
    return facto(n)//facto(p)//facto(n-p)
```

Solution de l'exercice IV.8 -

```
def binomial1(n, p):
    b = 1
    for k in range(p):
        b = b * (n-k) // (k+1)
    return b
```

Solution de l'exercice IV.9 -

```
def sommePuissances(n,p):
      """Entrées : deux entiers positifs
         Sortie  : la valeur de sum(k^p,k=1..n)"""
    s=0
    for k in range(1, n+1):
        s = s + k**p
    return s
```

```
for n in range(1, 11):
   print("La somme des cubes de 1 à {} vaut {}".format(n,
      sommePuissances(n,3)))
   print("La somme des entiers de 1 à {} au carré vaut {}".
      format(n,(sommePuissances(n,3))**2))
```

Solution de l'exercice IV.10 -

- ```
  def a(n):
      terme = 1
      for i in range(n):
          terme = terme*2/(i+1)
      return terme

  def S(n):
      somme = 0
      for k in range(n+1):
          somme = somme + a(k)
      return somme
  ```

  L'appel de `a(k)` effectue $2k$ opérations donc l'appel de `S(n)` effectue $C(n) = \sum_{k=0}^{n} 1 + 2k = n + 1 + 2\sum_{k=0}^{n} k = (n+1)^2$ opérations.
- On calcule a_n dans la boucle

  ```
  def S1(n):
      a = 1
      somme = 0
      for k in range(n+1):
          somme = somme + a
          a = a*2/(k+1)
      return somme
  ```

  On effectue maintenant $C_1(n) = \sum_{k=0}^{n} 3 = 3(n+1)$ opérations.

Solution de l'exercice IV.11 -

```
import math as m
def gamma(e):
    n = int(1/e) + 1
    h = 0
    for k in range(1, n+1):
        h = h + 1/k
    return h - m.log(n+1), h - m.log(n)
```

Solution de l'exercice IV.12 -

```
def muller(n):
    u = 5/2
    u_suivant = 17/5
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u
```

Dans la partie principale.

```
for k in range(50):
    print("u{} vaut {}".format(k, muller(k)))
```

La suite semble converger vers 25 alors qu'elle semblait s'approcher de 4 au début.
```

Solution de l'exercice IV.13 - Il suffit de changer les termes initiaux

```
def muller_frac(n):
    u = Fraction(5, 2)
    u_suivant = Fraction(17, 5)
    for i in range(n):
        u_avant = u
        u = u_suivant
        u_suivant = 30 - 129/u + 100/u/u_avant
    return u

for k in range(50):
    print("u{} vaut {}, {} sous forme décimale".format(k,
       muller_frac(k), float(muller_frac(k))))
```

Solution de l'exercice IV.14 -

```
def racines3(u):
    r = [0]*3
    v = u**(1/3)
    w = cmath.rect(1, 2*cmath.pi/3)
    for i in range(3):
        r[i] = v*w**i
    return r
```

Solution de l'exercice IV.15 -

```
def solutions1(p, q):
    delta  = q**2 + 4*p**3/27
    u = (-q + delta**(1/2))/2
    r = racines3(u)
    for i in range(3):
        r[i] = r[i] -p/(3*r[i])
    return r
```

Solution de l'exercice IV.16 -
$\left(u+\frac{a}{3}\right)^3 = u^3 + a.u^2 + \frac{a^2}{3}u + \frac{a^3}{27}$ or $u^3 + a.u^2 = -\_u - c$ car u est racine.
Ainsi $\left(u+\frac{a}{3}\right)^3 = \left(\frac{a^2}{3} - a\right)u + \frac{a^3}{27} - c = \left(\frac{a^2}{3} - b\right)\left(u+\frac{a}{3}\right) + \frac{a^3}{27} - c - \frac{a^3}{9} + \frac{a.b}{3}$.
$p = b - \frac{b^2}{3}$ et $q = c - \frac{a.b}{3} + \frac{2a^3}{27}$.

Solution de l'exercice IV.17 -

```
def solutions(a, b, c):
    p = b - a**2/3
    q = c - a*b/3 + 2*a**3/27
    d = -a/3
    if p == 0:
        r = racines3(-q)
    else:
        r = solutions1(p, q)
    for i in range(3):
        r[i] = r[i] + d
    return r
```

Solution de l'exercice IV.18 -

```
def carres(n, p):
    m = min(n, p)
    nb = 0
    for k in range(1, m + 1):
        nb = nb + (n + 1 - k)*(p + 1 - k)
    return nb

for i in range(1, 100):
    for j in range(i, 100):
        if carres(i, j) == 6400:
            print(i, j, i*j)
```

696

Solution de l'exercice IV.19 -

```
def longueur(n):
    a = n
    for i in range(n*n):
        if a == 1:
            return i
        elif a%2 == 0:
            a = a//2
        else:
            a = 3*a + 1

l_max = 1
n0 = 1
for n in range(2, 1500001):
    l = longueur(n)
    if l >= l_max:
        n0 = n
        l_max = l

print(n)
```

1117065 et 1126015, de longueur 528.

Solution de l'exercice IV.20 -

```
fin = 2020

d7 = 1
sens7 = 1
d8 = 1
sens8 = 1
somme = 0
for i in range(2, fin + 1):
    d7, sens7 = suivant(d7, sens7, 7)
    d8, sens8 = suivant(d8, sens8, 8)
    if d7 == 2 and d8 == 2:
        somme = somme + i
print(somme)
```

98882

Solution de l'exercice IV.21 -

```
from itertools import permutations

def test(suite):
    n = 0
    for k in range(1, 10):
        n = n*10 + suite[k-1]
        if n%k != 0:
            return False, 0
    return True, n

for x in permutations((1, 2, 3, 4, 5, 6, 7, 8, 9)):
    rep, n = test(x)
    if rep:
        print(n)
```

381654729

TP V

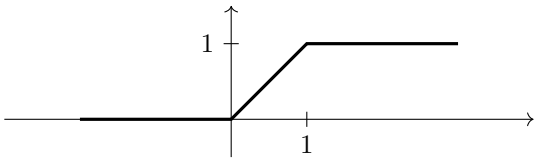
Instructions conditionnelles

Résumé *Dans ce T.P. nous allons utiliser les instructions conditionnelles* `if, else, elif`*. Elles permettent d'adapter le traitement des données selon les cas.*

1 Exercices

Exercice V.1

Écrire une fonction `seuil(x)` telle que $\text{seuil}(x) = \begin{cases} 0 & \text{pour } x \leqslant 0 \\ x & \text{pour } 0 \leqslant x \leqslant 1 \\ 1 & \text{pour } 1 \leqslant x \end{cases}$



Un année est dite bissextile si c'est un multiple de 4, sauf si c'est un multiple de 100. Toutefois, si c'est un multiple de 400, alors elle est considérée comme bissextile.

Exercice V.2 — Année bissextile

Écrire une fonction `bissextile(annee)` qui renvoie `True` ou `False` selon que l'année donnée en entrée sous forme d'un entier est ou n'est pas bissextile.
Contrairement à l'exercice III.11 on peut utiliser les instructions conditionnelles.

Exercice V.3 — Jours dans le mois

Écrire une fonction `jours(mois, annee)` qui renvoie le nombre de jours du mois donné par son numéro (entre 1 et 12) ; le paramètre année sert à déterminer le nombre de jours de février (mois 2) selon que l'année est ou n'est pas bissextile.

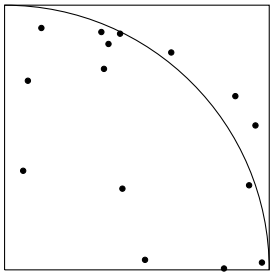
Exercice V.4 — Second degré

Écrire une fonction `racinesReelles(a, b, c)` qui, étant donné un polynôme de degré 2 $aX^2 + bX + c$ identifié par ses 3 coefficients réels, renvoie un triplet

- `(2, r1, r2)` si le polynôme admet deux racines réelles r_1 et r_2,
- `(1, r, r)` si le polynôme admet une racine double r,
- `(0, 0, 0)` si le polynôme n'admet pas de racine.

Exercice V.5 — Approximation
Écrire une fonction `approximation(x, n)` qui envoie un couple d'entiers `(a, b)` tel que $\frac{a}{b}$ est la meilleure approximation de x sous forme d'une fraction de dénominateur inférieur ou égal à n.
Par exemple `approximation(math.sqrt(2), 50)` renverra `(41, 29)`
et `approximation(-math.pi, 50)` renverra `(-22, 7)`.
Rappel : la fonction `round` permet de déterminer l'entier le plus proche d'un flottant.

Exercice V.6 — Méthode de Monte-Carlo
La méthode de Monte-Carlo permet d'approcher l'aire d'une surface incluse dans un rectangle. Pour cela on choisit n points aléatoirement dans le rectangle et on note la proportion de ceux qui sont dans la surface : r. L'aire de la surface sera approchée par $r.S$ où S est l'aire du rectangle. Dans le cas d'un quart-de-cercle inclus dans le carré unité $[0;1] \times [0;1]$ on calcule ainsi une valeur approchée de $\frac{\pi}{4}$.
Dans l'exemple ci-contre on peut approcher π par $4.\frac{12}{15} = 3,2$
La fonction `random()` du module `random` renvoie une valeur aléatoirement choisie entre 0 et 1.



Écrire une fonction `approx_pi(n)` qui utilise cette méthode pour déterminer une valeur approchée de π en choisissant n points dans le carré unité.

```
>>> approx_pi(1000000)
3.140632
```

Exercice V.7 — Triangles rectangles
On cherche les triangles rectangles dont les longueurs des cotés sont des entiers.
Plus précisément on cherche les entiers a, b et c tels que $a \leqslant b$ et $a^2 + b^2 = c^2$.
On dit que (a, b, c) est un **triplet pythagoricien**.
Écrire une fonction `pythagoriciens(n)` qui renvoie le nombre de triplets pythagoriciens (a, b, c) avec $1 \leqslant a \leqslant n$, $1 \leqslant b \leqslant n$ et $1 \leqslant c \leqslant n$.
Il y a 20 solutions pour $n = 50$, 127 pour $n = 200$ et 881 pour $n = 1000$.

Exercice V.8 — Suite $3n + 1$
On définit une suite $(u_n)_{n\in\mathbb{N}}$ par son premier terme p et par la relation

$$u_{n+1} = \begin{cases} u_n/2 & \text{si } n \text{ est pair} \\ 3u_n + 1 & \text{si } n \text{ est impair} \end{cases}$$

Écrire une fonction `suite(u0, n)` qui calcule u_n.

Exercice V.9 — PGCD
Proposer une fonction `pgcd(n, p)` qui détermine le plus grand diviseur commun de n et p entiers positifs, c'est-à-dire le plus grand entier qui divise n et p. Il est compris entre 1 et `min(n, p)`.

Exercice V.10 — Nombres premiers
Un entier $n \geqslant 2$ est **premier** s'il n'admet pas d'autre diviseur que 1 et lui-même.
On rappelle qu'un entier est premier si et seulement si il n'admet aucun diviseur entre 2 et $\lfloor\sqrt{n}\rfloor$.
Écrire une fonction `estPremier(n)` qui renvoie `True` et `False` selon que n est premier ou non.

Exercice V.11 — Conjecture de Golbach
Écrire une fonction `goldbach(n)` qui retourne un couple de nombres premiers (a, b) tels que $a+b = n$ lorsque n pair.
Exemple : `goldbach(232) -> (3, 229)` et `goldbach(252) -> (11, 241)`.
On peut exclure les cas où n est impair avec l'instruction

```
assert(n%2 == 0)
```

2 Compléments

2.1 Bataille navale

On considère un jeu de bataille navale simplifié : il ne s'agit pas de couler le porte-avions mais plutôt une barque tenant sur une seule case repérée par ses coordonnées `ligne0` et `colonne0` qui sont des variables globales. On fait un tir sur une case repérée par ses coordonnées `ligne` et `colonne`. On veut alors le comportement suivant :

- si la barque est exactement sur la case considérée, le programme imprime le texte "**Coulé**",
- si le tir atteint la bonne ligne ou la bonne colonne, le programme imprime "**En vue**",
- si le tir est totalement raté, le programme imprime "**À l'eau**".

On propose la fonction suivante :

```
def bataille(ligne,colonne):
    if ligne == ligne0 or colonne == colonne0:
        print("En vue")
    elif ligne == ligne0 and colonne == colonne0:
        print("Coulé")
    else :
        print("À l'eau")
```

Exercice V.12 — Correction
Pourquoi cette fonction n'est-elle pas correcte ?
Proposer une fonction valide.

Exercice V.13 — Amélioration
En général, à la bataille navale, un bateau n'est "en vue" que si la case visée est immédiatement voisine de celle du bateau. Modifier le programme de bataille navale afin de tenir compte de cette règle. On commencera par le cas où les cases diagonalement adjacentes de la barque sont "en vue" puis on traitera le cas où elle ne le sont pas.

2.2 Calcul mental

La fonction `randint` du module `random` permet de calculer un entier aléatoirement : `randint(1, 6)` renvoie un entier au hasard dans $\{1, 2, 3, 4, 5, 6\}$.

```
from random import randint
```

La fonction `sleep` du module `time` engendre une pause dans le programme : `sleep(5)` attend 5 secondes.

```
from time import sleep
```

Tester le programme suivant :

```
def addition1():
    a = randint(1,10)
    b = randint(1,10)
    print(a,'+',b,'=')
    time.sleep(4)
    return a+b
```

Exercice V.14 — Opérations
Construire une fonction `calculMental` qui propose de chercher au hasard une addition de nombres entiers à 3 chiffres ou un produit de nombres entiers à 2 chiffres et laisse un temps raisonnable pour la recherche de la réponse.

Exercice V.15 — Un jeu
Construire une fonction `challenge(n)` qui propose n lancements successifs de `calculMental`.

3 Solutions

Solution de l'exercice V.1 -

```
def seuil(x):
    if x < 0:
        return 0
    elif x < 1:
        return x
    else:
        return 1
```

Solution de l'exercice V.2 - On suit la définition en commençant par les exceptions

```
def bissextile(annee):
    if annee%400 == 0:
        return True
    elif annee%100 == 0:
        return False
    elif annee%4 == 0
        return True
    else:
        return False
```

On peut comprendre la règle sous la forme : les années bissextile sont celles multiples de 4 mais non multiple de 100 ou les années multiples de 400.

```
def bissextile(annee):
    return (annee%4 == 0 and not(annee%100 == 0)) or (annee
       %400 == 0)
```

Solution de l'exercice V.3 -

```
def jours(mois, annee):
    if mois == 2:
        if bissextile(annee):
            return 29
        else:
            return 28
    elif mois == 4 or mois == 6 or mois == 9 or mois == 11:
        return 30
    else:
        return 31
```

Solution de l'exercice V.4 -

```
def racinesReelles(a, b, c):
    """Entrees : 3 coefficients du polynome aX**2 + bX + c
       Sortie : le nombre de racines reelles et celles-ci"""
    delta = b**2 - 4*a*c
    if delta > 0:
        return (2, (-b + delta)/(2*a), (-b = delta)/(2*a))
    elif delta == 0:
        return (1, -b/(2*a), -b/(2*a))
    else:
        return (0, 0, 0)
```

Solution de l'exercice V.5 -

```
def approximation(x, n):
      """Entrées : un réel x et un entier positif n
         Sortie  : un couple (a,b) tel que a/b est la
             meilleure
                    approximation de x avec 1<=b<=n"""
    approx = (round(x), 1) # On initialise avec le dé
       nominateur 1
    ecart = abs(x - round(x))
    for denom in range(2, n+1):
        num = round(x*denom)
        if abs(x - num/denom) < ecart:
            ecart = abs(x - num/denom)
            approx = (num, denom)
    return approx
```

Solution de l'exercice V.6 -

```
from random import random

def approx_pi(n):
    dedans = 0
    for i in range(n):
        x = random()
        y = random()
        if x**2 + y**2 < 1:
            dedans = dedans + 1
    return dedans/n*4
```

Solution de l'exercice V.7 -

```
def pythagoriciens(n):
    nombre = 0
    for a in range(1, n+1):
        for b in range(i, n+1):
            for c in range(j, n+1):
                if a**2 + b**2 == c**2:
                    nombre = nombre + 1
    return nombre
```

Pour $n = 1000$, ce programme est trop lent.
On peut obtenir la valeur avec

```
def pythagoriciens1(n):
    nombre = 0
    for a in range(1, n+1):
        for b in range(a, n+1):
            r = (a**2 + b**2)**0.5
            if r == int(r) and r <= n:
                    nombre = nombre + 1
    return nombre
```

Solution de l'exercice V.8 -

```
def suite(u0, n):
    u = u0
    for i in range(n):
        if n%2 == 0!
            u = u//2
        else:
            u = 3*u + 1
    return u
```

Solution de l'exercice V.9 -

```
def pgcd(a,b):
    commun = 1
    for k in range(1, min(a,b)+1):
        if (a % k == 0) and (b % k == 0 ):
            commun = k
    return commun
```

Solution de l'exercice V.10 -

```
def estPremier(n):
    max = int(n**0.5)
    for k in range(2, max+1):
        if n%k == 0:
            return False
    return True
```

Solution de l'exercice V.11 -

```
def goldbach(n):
    assert(n%2 == 0)
    for k in range(2, n//2+1):
        if estPremier(k) and estPremier(n-k):
            return (k, n-k)
```

Solution de l'exercice V.12 - Si on calcule la fonction avec les bonnes coordonnées, la condition `ligne == ligne0 or colonne == colonne0` va être évaluée en `True` donc la fonction va retourner "En vue" au lieu de "Coulé".
Il suffit d'inverser les deux premières conditions.

```
def bataille(ligne,colonne):
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif ligne == ligne0 or colonne == colonne0:
        return "En vue"
    else :
        return "À l'eau"
```

Solution de l'exercice V.13 - Si les points proches contiennent la diagonale alors ils forment un carré.

```
def bataille2(ligne,colonne):
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif     abs(ligne - ligne0) <= 1
         and abs(colonne - colonne0) <= 1:
        return "En vue"
    else:
        return "À l'eau"
```

Sinon il faut une égalité d'une coordonnée et un écart de 1 au plus pour l'autre donc une somme des écarts majorée par 1.

```
def bataille2(ligne,colonne):ff
    if ligne == ligne0 and colonne == colonne0:
        return "Coulé"
    elif  (abs(ligne-ligne0) + abs(colonne-colonne0) <= 1):
        return "En vue"
    else :
        return "À l'eau"
```

Solution de l'exercice V.14 -

```
from random import randint
import time

def calculMental():
    op = randint(1, 2)
    if op == 1:
        a= randint(100, 999)
        b= randint(100, 999)
        print("{} + {} = ?".format(a, b))
        time.sleep(6)
        print(a + b)
    else:
        a = randint(10, 99)
        b = (10, 99)
        print("{} * {} = ?".format(a, b))
        time.sleep(13)
        print(a*b)
```

Solution de l'exercice V.15 -

```
def challenge(n):
    for i in range(n):
    calculMental()
```

TP VI

LISTES

Dans ce T.P. nous allons utiliser les listes comme variables, leur construction fera l'objet du sujet suivant.
La structure des fonctions à écrire sera souvent du même type :

```
def une-fonction(var1, ..., varp, une_liste):
    n = len(une_liste)
    # Initialisations
    for i in range(n):
        # traitement pour chaque i
    # traitement global
    return le_resultat
```

Quand on n'a pas besoin de l'indice, Python permet la construction

```
def une-fonction(var1, ..., varp, une_liste):
    # Initialisations
    for x in une_liste:
        # traitement pour chaque x
    # traitement global
    return le_resultat
```

Plusieurs listes exemples sont données dans le fichier `TP06_ex.py`

1 Statistiques

Exercice VI.1 — Moyenne
Écrire une fonction `moyenne(X)` *qui calcule la moyenne des termes de la liste* X *:*
`moyenne(X)` $= \dfrac{1}{n}\displaystyle\sum_{i=0}^{n}$ `X[i]` *avec* $n =$ `len(X)`.
`moyenne(X0)` *donne* `1.9956097560975619`

La **variance** d'une liste est définie par $\text{var}(X) = \dfrac{1}{n}\displaystyle\sum_{i=0}^{n}\big(\texttt{X[i]} - m_X\big)^2$ avec m_X égal à la moyenne de la liste et $n =$ `len(X)`.

Exercice VI.2 — Variance
Écrire une fonction `var(X)` *qui calcule la variance de la liste* X.
On affectera la moyenne dans une variable avant la boucle pour éviter de répéter le même calcul.

`variance(Z0)` *donne* `6.558001189767997`

Si X et Y sont deux listes de même longueur, la **covariance** de X est Y est

$\text{cov}(X,Y) = \frac{1}{n}\sum_{i=0}^{n}\big(\texttt{X[i]} - m_X\big)\big(\texttt{Y[i]} - m_Y\big)$ avec m_X égal à la moyenne de la liste X, m_Y égal à la moyenne de la liste Y et n est la longueur commune de X et de Y.
On pourra noter que $\text{var}(X) = \text{cov}(X,X)$

Exercice VI.3 — Covariance

Écrire une fonction `cov(X, Y)` *qui calcule la covariance des listes X et Y.*

Il sera nécessaire ici d'accéder aux éléments de `X` *et* `Y` *par leurs indices.*

`cov(T0, Y0)` *donne* `-1.6044390243902438`

On admet que, X et Y sont deux listes de même longueur, alors la **droite de régression** approchant Y par rapport à X est la droite d'équation $y = ax + b$ avec $m_Y = a.m_X + b$ et $a.\text{var}(X) = \text{cov}(X,Y)$.

Exercice VI.4 — Droite de régression

Écrire une fonction `regression(X, Y)` *qui calcule les coefficientsa et b de la droite de régression de X à Y.*

`regression(X0, Y0)` *donne* `(2.2937900415993773, 1.7193194828375322)`

1.1 Premiers graphes

Le module `matplotlib` permet de tracer des représentations de valeurs, c'est particulièrement utile dans le cas de listes. Nous utiliserons la sous-bibliothèque `pyplot` que l'on simplifiera par `plt`.

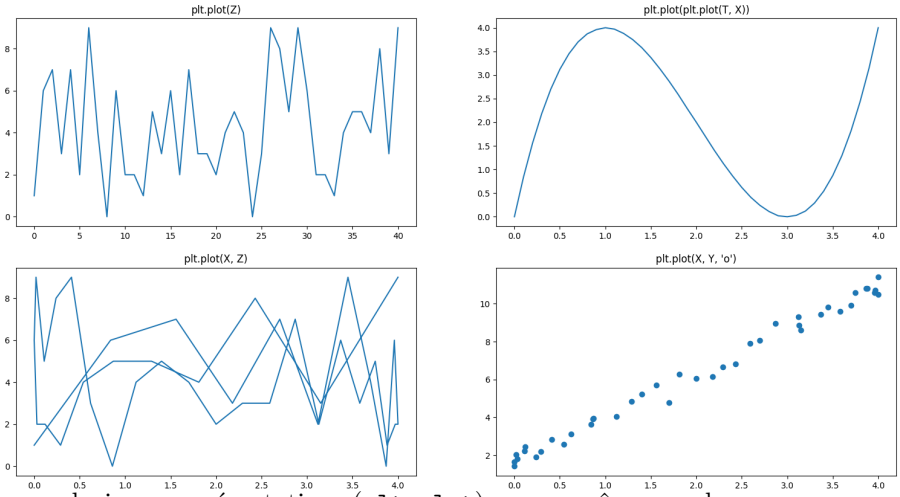
```
import matplotlib.pyplot as plt
```

Il existe deux modes de tracé.

1. Le mode standard construit les graphismes **sans les afficher**, ils sont enrichis pas-à-pas. Quand toutes les instructions sont écrites, on demande l'affichage par `plt.show()` ; une fenêtre avec les graphismes apparaît et il faut la refermer pour continuer à utiliser Python. Nous utiliserons ce mode.
2. Le mode interactif permet l'affichage de chaque commande en temps réel, sans monopoliser Python. Le principal inconvénient est que les graphismes deviennent vite surchargés, il faut régulièrement les effacer (`plt.clf()`).
3. On passe en mode interactif avec `plt.ion`, on en sort avec `plt.ioff`.

La fonction principale est `plt.plot`. Elle admet 2 utilisations.

1. `plt.plot(X)` où `X` est une liste, relie les points de coordonnées `(i, X[i])` par des segments de droite, elle présente les valeurs de `X` dans l'ordre de la liste.
2. `plt.plot(X, Y)` où `X` et `Y` sont deux listes **de même longueur** relie les points de coordonnées `(X[i], Y[i])` par des segments de droite, elle présente les évolutions des les valeurs de `Y` en fonction de celles de `X`.
3. Quand la liste n'est pas triée dans l'ordre croissant, la représentation ci-dessus n'est pas très signifiante. Parmi les nombreuses options de tracé, il existe la possibilité de représenter simplement les points sans tracer les segments avec un paramètre supplémentaire `'o'`,



On peut tracer plusieurs représentations (`plt.plot`) sur un même graphe.

Exercice VI.5 — Tracé de la droite de régression

Après avoir calculé les coefficients (a et b) de la droite de régression approchant `Y0` *en fonction de* `X0`*, représenter sur un même graphe les points de coordonnées* `(X0[i], Y0[i])` *et la droite de régression. Pour tracer une droite d'équation* $y = ax + b$ *sur* $[\alpha, \beta]$*, on pourra tracer le segment entre les points* $(\alpha, a.\alpha + b)$ *et* $(\beta, a.\beta + b)$*.*

2 Tests

Lorsque l'on effectue un test sur une liste, on maintient une variable à valeur booléenne que l'on initialise avec une valeur qui est, en général, le résultat pour une liste vide. Par exemple, si on cherche si au moins un terme d'une liste est nul, le résultat initial est `False` : on n'a pas encore trouve de terme nul. On parcourt ensuite la liste et on transforme le résultat en `True` si on trouve un terme nul. Pour l'instant on ne cherchera pas à cesser la recherche dès qu'on a trouvé un terme nul.

```
def contient0(liste):
    out = False
    for x in liste:
        if x == 0:
            out = True
    return out
```

On ne cherchera pas pour l'instant à optimiser les calculs et à renvoyer un résultat dès que le résultat est certain.

Exercice VI.6 — Positivité

Tester la positivité d'une liste est ambigu : on peut tester s'il existe un élément positif ou si tous les éléments sont positifs.
Ici positif signifie positif ou nul.

1. *Écrire une fonction* `tout_positif(liste)` *qui renvoie* `True` *ou* `False` *selon que tous les éléments de la liste sont positifs ou non.*
2. *Écrire une fonction* `existe_positif(liste)` *qui renvoie* `True` *s'il existe au moins un élément positif dans la liste et* `False` *sinon.*

Exercice VI.7 — Croissance

Écrire une fonction `croissante(liste)` *qui renvoie* `True` *ou* `False` *selon que les termes de la liste forme une suite croissante ou non.*
`croissance(T0)` *donne* `True`, `croissance(X0)` *donne* `False`

3 Maximums

La recherche du maximum d'une expression dépendant des termes d'une liste doit être initialisée avec soin ; par exemple, il ne faut pas initialiser le maximum des termes d'une liste par 0 et rechercher les termes qui dépassent car il se peut que tous les termes soient négatifs. L'initialisation d'un maximum doit être une valeur possible de l'expression.

```
def maximum(liste):
    maxi = liste[0]
    for x in liste:
        if x > maxi:
            maxi = x
    return maxi
```

Exercice VI.8 — Minimum
Écrire une fonction `minimum(liste)` *qui calcule le minimum d'une liste non vide.*
`minimum(Y0)` *donne* `1.42`

Exercice VI.9 — Indice du maximum
Écrire une fonction `indiceMax(liste)` *qui calcule un indice en lequel une liste non vide atteint son maximum. Si la liste atteint son maximum plusieurs fois, quel indice votre fonction renvoie-t-elle ?*
`indiceMax(Z0)` *donne* `6` *ou* `40`.

Exercice VI.10 — Sommes de 3 termes
Écrire une fonction `maximum3(liste)` *qui calcule le maximum de la somme de 3 termes consécutifs d'une liste de longueur au moins 3.*
`maximum3(X0)` *donne* `11.93`

Exercice VI.11 — Deuxième valeur
Écrire une fonction `second(liste)` *qui calcule le deuxième plus grand terme d'une liste de nombres. En cas d'ex-æquo pour le maximum, la fonction peut renvoyer le maximum.*
`second(Y0)` *donne* `10.8`, `second(Z0)` *donne* `9`

L'augmentation maximale d'une liste est le plus grand accroissement `liste[j] - liste[i]` pour $i \leqslant j$. Sa valeur est toujours positive et vaut 0 dans le cas d'une liste décroissante. L'augmentation maximale est donc, en notant L_i la valeur de `liste[i]` et n est la longueur de la liste,

$$\text{augMax} = \max\{L_j - L_i \ ; \ 0 \leqslant i \leqslant j < n\}$$

Exercice VI.12 — Augmentation maximale
Écrire une fonction `augMax(liste)` *qui calcule l'augmentation maximale d'une liste de nombres.*
`augMax(Y0)` *donne* `9.76`

Exercice VI.13 ** **— En plus**

1. *Écrire une fonction* `second(liste)` *qui calcule la deuxième valeur d'une liste, distincte du maximum, sauf dans le cas d'une liste constante.*
2. *Écrire une fonction* `majoritaire(liste)` *qui calcule l'élément majoritaire d'une liste* **triée** *avec une unique boucle* `for`*. Que compte la fonction si la liste n'est pas triée ?*
3. *Écrire une fonction* `pivot(liste)` *qui renvoie un indice i tel que* $\left|\sum_{k=0}^{i-1} L_k - \sum_{k=i}^{n-1} L_k\right|$ *est minimale : on veut couper la liste en deux parties de sommes presqu'égales.*
4. *Écrire une fonction* `augMax(liste)` *qui n'utilise qu'une boucle* `for`*.*
5. *Écrire une fonction qui calcule la somme maximale d'une tranche de liste (suite de termes consécutifs) en n'employant qu'une boucle* `for`*.*

4 Solutions

Solution de l'exercice VI.1 -

```
def moyenne(X):
    n = len(X)
    somme = 0
    for x in X:
        somme = somme + x
    return somme/n
```

Solution de l'exercice VI.2 -

```
def var(X):
    n = len(X)
    mX = moyenne(X)
    somme = 0
    for x in X:
        somme = somme + (x - mX)**2
    return somme/n
```

Solution de l'exercice VI.3 -

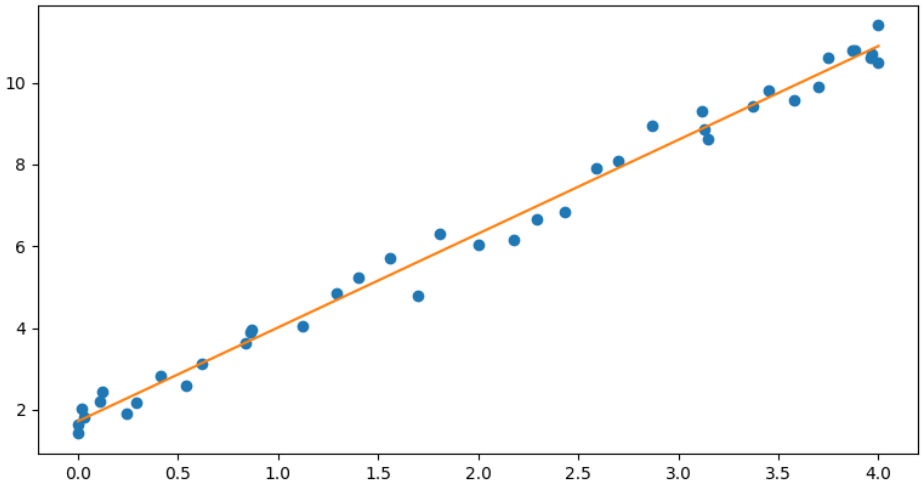
```
def cov(X, Y):
    n = len(X)
    mX = moyenne(X)
    mY = moyenne(Y)
    somme = 0
    for i in range(n):
        somme = somme + (X[i] - mX)*(Y[i] - mY)
    return somme/n
```

Solution de l'exercice VI.4 -

```
def regression(X, Y):
    a = cov(X, Y)/var(X)
    b = moyenne(Y) - a*moyenne(X)
    return a, b
```

Solution de l'exercice VI.5 -

```
a, b = regression(X0, Y0)
c = a*4 + b
plt.plot(X0, Y0, 'o')
plt.plot([0, 4], [b, c])
plt.show()
```



Solution de l'exercice VI.6 - L'initialisation est différente.

1. Si on cherche à savoir si tous les éléments sont positifs la réponse est `True` jusqu'à ce qu'on trouve un élément négatif. Il est important de ne pas remettre le résultat à `True` si on retrouve une valeur positive, il n'y a pas de traitement `else`.

```
def tout_positif(liste):
                False sinon"""
    resultat = True
    for x in liste:
        if x < 0:
            resultat = False
    return resultat
```

2. Ici la réponse est `False` jusqu'à preuve du contraire.

```
def existe_positif(liste):
    resultat = False
    for x in liste:
        if x >= 0:
            resultat = True
    return resultat
```

Solution de l'exercice VI.7 -

```
def croissante(liste):
    n = len(liste)
    resultat = True
    for i in range(n-1):
        if liste[i] > liste[i+1]:
            resultat = False
    return resultat
```

Solution de l'exercice VI.8 -

```
def minimum(liste):
    mini = liste[0]
    for x in liste:
        if x < mini:
            mini = x
    return mini
```

Solution de l'exercice VI.9 -

```
def indiceMax(liste):
    n = len(liste)
    i_maxi = 0
    for i in range(1, n):
        if liste[i] > liste[i_max]:
            i_max = i
    return i_max
```

L'indice renvoyé est le premier en lequel la liste atteint son maximum.
Si on remplace le test par `if liste[i] >= liste[i_max]:`, ce sera le dernier indice.

Solution de l'exercice VI.10 -

```
def maximum3(liste):
    n = len(liste):
    max3 = liste[0] + liste[1] + liste[2]
    for i in range(1, n-2):
        s3 = liste[i] + liste[i+1] + liste[i+2]
        if s3 > max3:
            max3 = s3
    return max3
```

Solution de l'exercice VI.11 -

```
def second(liste):
    n = len(liste)
    max1 = max(liste[0], liste[1])
    max2 = mini(liste[0], liste[1])
    for i in range(2, n):
        x = liste[i]
        if x > max1:
            max2 = max1
            max1 = x
        elif x > max2:
            max2 = x
    return max2
```

Solution de l'exercice VI.12 - On ne recalcule pas `liste[i] - liste[i]`

```
def augMax(liste):
    n = len(liste)
    augM = 0
    for i in range(n-1):
        for j in range(i, n):
            aug = liste[j] - liste[i]
            if aug > augM:
```

```
                augM = aug
    return augM
```

Solution de l'exercice VI.13 -

1. Il faut gérer les cas d'égalité.

```
def second(liste):
    n = len(liste)
    max1 = liste[0]
    max2 = liste[0]
    for i in range(1, n):
        x = liste[i]
        if x > max1:
            max2 = max1
            max1 = x
        elif (max1 == max2 > x) or (max1 > x > max2):
            max2 = x
    return max2
```

2. Si la liste n'est pas triée, on renvoie l'élément le plus répété.

```
def majoritaire(liste):
    n = len(liste)
    nb_maj = 1
    maj = liste[0]
    encours = 1
    for i in range(1, n):
        if liste[i] == liste[i-1]:
            encours += 1
            if encours > nb_maj:
                nb_maj = encours
                maj = liste[i]
        else:
            encours = 1
    return maj
```

3. Pour chaque i on calcule les deux sommes qu'il sépare. On peut le faire avec une addition et une soustraction à chaque étape. On notera qu'il y a $n+1$ résultats possibles : de 0 à n.

```
def pivot(liste):
    n = len(liste)
    gauche = 0
    droite = 0
    for x in liste:
        droite = droite + x
    e_min = abs(droite - gauche)
    i_min = 0
    for i in range(n):
        x= liste[i]
        gauche = gauche + x
        droite = droite - x
        e = abs(gauche - droite)
        if e < e_min:
           e_min = e
           i_min = i + 1
    return i_min
```

`pivot([1, 2, 3, 6])` donne `3`

4. On remarque que le maximum d'un écart qui finit en j est la différence entre L_j et le minimum jusqu'à j : il suffit donc de calculer ce minimum dans la boucle.

```
def augMax(liste):
    n = len(liste)
    augM = 0
    mini = liste[0]
    for i in range(1, n):
        mini = min(mini, liste[i])
        aug = liste[i] - mini
        if aug > augM:
            augM = aug
    return augM
```

5. C'est presque l'exercice précédent : on veut le minimum des $\sum_{k=i}^{j} L_k$ pour $0 \leqslant i \leqslant j < n$. Si on pose $S_i = \sum_{k=0}^{j} L_k$ et $S_{-1} = 0$, on veut le minimum des $S_j - S_{i-1}$.

```
def trancheMax(liste):
    n = len(liste)
    tMax = 0
    sMin = 0
    S = 0
    for i in range(n):
        S = S + liste[i]
        t = S - sMin
        if t > tMax:
            tMax = t
        sMin = min(sMin, S)
    return tMax
```

`trancheMax([2, -1, 3, -3, 2])` donne `4`

TP VII

Listes : 2

Dans ce T.P. nous allons utiliser construire ou modifier des listes. Dans le cas de constructions de listes, les énoncés ne considèrent que des listes de taille fixées à l'avance. Les cas plus général de listes sans connaître à l'avance la taille sera vu dans un autre T.P.
Une liste de taille n pourra être construite selon le motif

```
def fonction(...):
    ...
    L = [0]*n
    for i in range(n):
        ...
        L[i] = ...
    return L
```

1 Fonctions mathématiques

Exercice VII.1 — Carrés
Écrire une fonction `carre(liste)` *qui renvoie la liste de même taille que la liste passée en paramètre et dont les éléments sont les carrés de éléments de celle-ci.*

La fonction ci-dessus peut se généraliser.
Une fonction peut recevoir une autre fonction comme paramètre.

Exercice VII.2 — Image d'une liste par une fonction
Écrire une fonction `appliquer(f, liste)` *qui renvoie la liste des valeurs $f(x)$ pour les éléments x de la liste.*

Par exemple si on définitions

```
def harmo(x):
    return (x+4)/(x+1)
```

`appliquer(harmo, [0, 1, 2, 3, 4])` renverra `[4.0, 2.5, 2.0, 1.75, 1.6]`.

On peut écrire le résultat de `appliquer(f, liste)` plus simplement avec

```
[f(x) for x in liste]
```

1.1 Graphe d'une fonction

Exercice VII.3
Écrire une fonction `listeX(a, b, n)` *qui calcule une liste de longueur* n *dont les valeurs sont équiréparties entre* a *et* b.

`listeT(1, 3, 5)` doit renvoyer `[1.0, 1.5, 2.0, 2.5, 3.0]`.
On remarquera que l'écart entre deux points consécutifs est $\frac{b-a}{n-1}$, dans la liste renvoyée le terme d'indice i est donc $a + i\frac{b-a}{n-1}$. Il est recommandé de stocker la valeur $\frac{b-a}{n-1}$ dans une variable.

On peut maintenant représenter une fonction sur un intervalle $[a; b]$.

1. On définit la fonction python correspondant à la fonction à étudier.
   On pourra utiliser le module `math` pour obtenir les fonctions mathématiques usuelles.

   ```
   import math as m
   ```

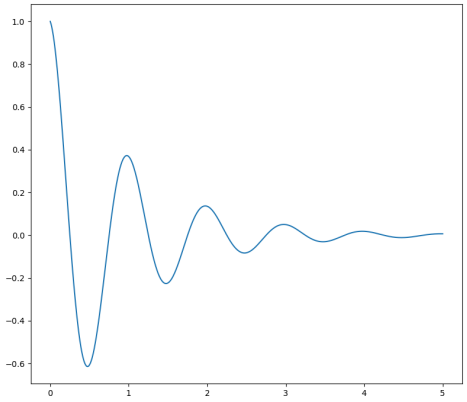
   On pourra utiliser `m.cos`, `m.exp`, `m.pi`, ...
2. On crée une liste d'abscisse avec `X = listeX(a, b, 500)` [1].
3. On crée la liste des ordonnées avec `Y = appliquer(f, X)`
4. On trace le graphe, il faudra avoir chargé le sous-module `pyplot`.

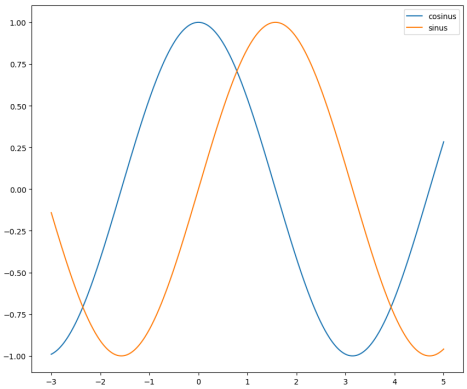
   ```
   import matplotlib.pyplot as plt
   ...
   plt.plot(X, Y)
   plt.show()
   ```

Exercice VII.4
Tracer le graphe de $f : x \mapsto e^{-x}\cos(\pi x)$ *sur* $[0 : 5]$.

Quand on veut tracer les graphes de plusieurs fonctions, on peut les référencer en ajoutant la variable optionnelle `label` qui qui doit être une chaîne de caractère, elle sera affichée par `plt.legend()`.

```
X = listeX(-3, 5, 500)
Y1 = appliquer(m.cos, X)
Y2 = appliquer(m.sin, X)
plt.plot(X, Y1, label="cosinus")
plt.plot(X, Y2, label="sinus")
plt.legend()
plt.show()
```





1. 500 points suffisent dans la plupart des cas.

2 Tirages aléatoires

Dans le module `random`, la fonction à deux variables entières `randint(a, b)` renvoie un entier au hasard compris entre a et b avec a et b compris.

```
from random import randint
print(randint(4, 12))
```

On obtient un entier différent à chaque appel appartenant à $\{4, 5, \dots, 12\}$.

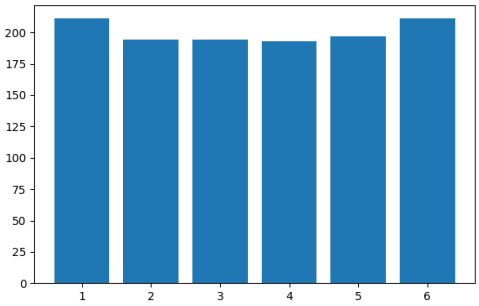
Exercice VII.5 — Tirages de dés
Écrire une fonction `tiragesDe(n)` *qui renvoie une liste de taille n dont les valeurs sont choisies aléatoirement entre 1 et 6.*
Par exemple `tiragesDe(12)` *peut renvoyer* `[6, 5, 3, 3, 2, 2, 6, 4, 6, 5, 6, 1]`.

Exercice VII.6 — Occurrences
Écrire une fonction `occurrences(liste)` *qui renvoie une liste de taille 6 dont les valeurs sont les nombres d'occurrences des entiers de 1 à 6 dans une liste ne contenant que des entiers entre 1 et 6. Le terme d'indice i de la liste ($0 \leqslant i < 6$) contiendra le nombre d'occurrences de $i+1$.*

On peut représenter le résultat par un histogramme.

1. On crée une liste de tirages : `T = tiragesDe(1200)`.
2. On répartit les tirages : `occ = occurrences(X)`.
3. On crée la liste des valeurs comptées : `X = [1, 2, 3, 4, 5, 6]`
4. On trace le graphe en barres

```
plt.bar(X, occ)
plt.show()
```



3 Suites

On a déjà vu la suite des nombres de Fibonacci ils sont définis par

$$F_0 = 0,\ F_1 = 1 \text{ et } F_{n+2} = F_{n+1} + F_n \text{ pour } n \in \mathbb{N}$$

Exercice VII.7 — Nombres de Fibonacci
Écrire une fonction `liste_fibo(n)` *qui renvoie la liste formée des $n+1$ premiers nombres de Fibonacci, de F_0 à F_n. On notera qu'on doit renvoyer $n+1$ valeurs.*

Les nombres de Catalan sont des entiers qui apparaissent dans de nombreux problèmes de dénombrement. Une de leurs définitions est la récurrence :

$$C_0 = 1 \text{ et } C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i} \text{ pour } n \in \mathbb{N}$$

Exercice VII.8 — Nombres de Catalan
Écrire une fonction `catalan(n)` *qui renvoie la liste formée des nombres de Catalan de C_0 à C_n.*

On rappelle qu'on a $\binom{n}{k+1} = \dfrac{n-k}{k+1}\binom{n}{k}$.

Exercice VII.9 — Coefficients binomiaux, bis
Écrire une fonction `liste_binom(n)` qui renvoie la liste des $\binom{n}{k}$ pour $0 \leqslant k \leqslant n$.

4 Transformations de listes

Exercice VII.10 — Échanges
Écrire une fonction `echanger(liste, i, j)` qui échange les termes d'indices i et j dans la liste. Cette fonction ne renvoie rien, elle doit modifier la liste.

Si on veut retourner une liste, on peut le faire en créant une nouvelle liste. Le plus simple alors est d'en extraire les termes "à l'envers" : `etsil = liste[ : : -1]`.
Dans ce qui suit on voudra effectuer les modifications en interne,
on parle de transformation **en place**.

Exercice VII.11 — Retourner une liste
Écrire une fonction `retourner(liste)` *qui retourne une liste en place.*

On se pose maintenant le problème de décaler une liste avec rotation vers la droite : les derniers éléments reviennent en tête.
Par exemple un décalage d'une place de `[4, 2, 8, 7, 5, 2, 1]` doit être `[1, 4, 2, 8, 7, 5, 2]`

Exercice VII.12 — Décaler d'une place
Écrire une fonction `decaler1(liste)` *qui décale une liste d'une position en place.*

Exercice VII.13 — Décalage
En déduire une fonction `decaler(k, liste)` *qui décale une liste de k positions vers la droite.*

Le décalage suggéré ci-dessus demande beaucoup d'affectations.
Il existe une astuce qui effectue moins de calculs. On va l'illustrer sur l'exemple ci-dessus.

1. Le décalage de 3 places de `[1, 4, 2, 8, 7, 5, 2]` est `[7, 5, 2, 1, 4, 2, 8]`. On voit que les 3 derniers termes sont en tête et les $7-3$ premiers termes sont maintenant à la fin.
2. Si on retourne la liste on arrive à `[2, 5, 7, 8, 2, 4, 1]`, on voit que les 3 derniers termes sont maintenant en tête mais ils sont à l'envers, de même pour les $7-3$ premiers termes arrivés à la fin et retournés.
3. On retourne alors la portion des 3 premiers termes, on arrive à `[7, 5, 2, 8, 2, 4, 1]`.
4. On retourne les $7-3$ derniers termes, on aboutit à la liste décalée.

Exercice VII.14 — Retournement partiel
Écrire une fonction `inverser(liste, i, j)` *qui retourne en place les éléments de la liste compris entre les position i et j, borne i comprise et borne j exclue.*

Exercice VII.15 — Décalage
En déduire une fonction `decaler(k, liste)` *qui décale une liste de k positions vers la droite.*

5 En plus

Exercice VII.16 — Drapeau polonais
Une liste ne contient que les valeurs 0 et 1. Écrire une fonction qui transforme la liste en plaçant tous 0 avant les 1 (on obtient 2 bandes, comme le drapeau polonais).

Si on remplace `liste[i] == 0` par `liste[i] < p` et `liste[i] == 1` par `liste[i] >= p`, on obtient un algorithme de séparation qui sera utile dans le tri rapide.

Exercice VII.17 — Drapeau hollandais
Une liste ne contient que les valeurs -1, 0 et 1. Écrire une fonction qui transforme la liste en plaçant les termes dans l'ordre. On obtient 3 bandes, comme le drapeau néerlandais.Le concepteur de cet algorithme est Edsger Dijkstra, qui est hollandais.

Cet algorithme permet de séparer selon un pivot p : les termes strictement inférieur à p, les termes égaux à p et les termes strictement supérieur à p.

6 Solutions

Solution de l'exercice VII.1 -

```
def carres(liste):
    n = len(liste)
    L = [0]*n
    for i in range(n):
        L[i] = liste[i]**2
    return L
```

Solution de l'exercice VII.2 -

```
def appliquer(f, liste):
    n = len(liste)
    L = [0]*n
    for i in range(n):
        L[i] = f(liste[i])
    return L
```

Solution de l'exercice VII.3 -

```
def listeX(a, b, n):
    pas = (b-a)/(n-1)
    X = [0]*n
    for k in range(n):
        X[k] = a + k*pas
    return X
```

Solution de l'exercice VII.4 -

```
def f(x):
    return m.exp(-x)*m/cos(m.pi*x)

X = listeX(0, 5, 500)
Y = appliquer(f, X)
plt.plot(X, Y)
plt.show()
```

Solution de l'exercice VII.5 -

```
from random import randint

def tiragesDe(n):
    """Entrée : un entier positif
       Sortie : la liste des tirages de n dés"""
    resultat = [0]*n
    for i in range(n):
        resultat[i] = randint(1, 6)
    return resultat
```

Solution de l'exercice VII.6 -

```
def occurrences(liste):
    """Entrée : un entier positif
       Sortie : la liste de la distribution des valeurs
                d'une liste d'entiers de 1 à 6"""
    n = len(liste)
    resultat = [0]*6
    for i in range(n):
        k = liste[i] - 1
        resultat[k] += 1
    return resultat
```

Solution de l'exercice VII.7 -

```
def liste_fibo(n)
    F = [0]*(n+1)
    F[1] = 1 # F[0] vaut déjà 0
    for k in range(n-1): #
        F[k+2] = F[k+1] + F[k]
    return F
```

Solution de l'exercice VII.8 -

```
def catalan(n):
    """Entrée : un entier positif
       Sortie : la liste des nombres de catalan de C(0) à C(n)
          """
    C = [0]*(n+1)
    C[0] = 1
    for k in range(n): # On calcule C(k+1)
        somme = 0
        for i in range(k+1): # i de 0 à k
            somme = somme + C[i]*C[k-i]
        C[k+1] = somme
    return C
```

Solution de l'exercice VII.9 -

```
def liste_binom(n):
    bin = [0]*n
    bin[0] = 1
    for k in range(n):
        bin[k+1 = bin[k]*(n-k)//(k+1)
    return bin
```

Solution de l'exercice VII.10 -

```
def echanger(liste, i, j):
    temp = liste[i]
    liste[i] = liste[j]
    liste[j] = temp
```

Solution de l'exercice VII.11 - On utilise la fonction `echanger`.
On échange les termes symétriques en s'arrêtant à la moitié de la longueur, sinon on revient à la liste initiale. On doit échanger les éléments d'indice 0 à $p-1$ avec leur symétrique si $n = 2p$ ou $n = 2p+1$ (le milieu reste en place), on s'arrête à $\left\lfloor \frac{n}{2} \right\rfloor - 1$.

```
def retourner(liste):
    n = len(liste)
    for i in range(n//2):
        echanger(liste, i, n-1-i)
```

Solution de l'exercice VII.12 - On place les éléments décalés depuis la fin. On doit conserver la valeur finale pour l'affecter à `liste[0]` à la fin.

```
def decaler1(liste):
    n = len(liste)
    temp = liste[-1]
    for i in range(n-1, 0, -1):
        liste[i] = liste[i-1]
    liste[0] = temp
```

Solution de l'exercice VII.13 -

```
def decaler(k, liste):
    for i in range(k):
        decaler1(liste)
```

Solution de l'exercice VII.14 - Si $j = i + 2p$, il y a un nombre pairs d'éléments à inverser : on échange i et $j-1$, $i+1$ et $j-2$, jusqu'à $i+p-1$ et $i+p$.
Si $j = i + 2p + 1$, il y a un nombre impairs d'éléments à inverser : on échange i et $j-1$, $i+1$ et $j-2$, jusqu'à $i+p-1$ et $i+p+1$. L'élément d'indice $i+p$ est laissé à sa place.
Dans les deux cas, la première coordonnée k varie dans `range(i, i+p)` ; on a $i + p = \left\lfloor \frac{i+j}{2} \right\rfloor$. La seconde coordonnée est $i + j - k - 1$.

```
def inverser(liste, i, j):
    m = (j + i)//2
    for k in range(i, m):
        echanger(liste, k, i + j - k -1)
```

Solution de l'exercice VII.15 -

```
def decaler(k, liste):
    n = len(liste)
    retourner(liste)
    inverser(liste, 0, k)
    inverser(liste, k, n)
```

Solution de l'exercice VII.16 -
En plus de l'indice de boucle i, on utilise une variable $k \leqslant i$ telle que :
les éléments d'indices entre 0 et $k-1$ sont 0
les éléments d'indices entre k et $i-1$ sont 1

```
def zero_un(liste):
    n = len(liste)
    k = 0
    for i in range(n):
        if liste[i] == 0:
            echanger(liste, k, i)
            k = k + 1
```

Solution de l'exercice VII.17 - Ici utilise deux variables, k_1 et k_2.
les éléments d'indices entre 0 et k_1-1 sont -1
les éléments d'indices entre k_1 et k_2-1 sont 0
les éléments d'indices entre k_2 et $i-1$ sont 1

```
def flag(liste):
    n = len(liste)
    k1 = 0
    k2 = 0
    for i in range(n):
        if liste[i] == -1:
            echanger(liste, i, k2)
            echanger(liste, k1, k2)
            k1 = k1 + 1
            k2 = k2 + 1
        if liste[i] == 0:
            echanger(liste, i, k2)
            k2 = k2 + 1
```

TP VIII

Boucles conditionnelles

Dans ce T.P. nous allons utiliser l'instruction de répétitions sous conditions, `while`. Bien que plus puissantes que les boucles `for`, les boucles `while` nécessitent un grand soin pour s'assurer de la sortie de la boucle, elles ne seront utilisées que lorsqu'elles sont nécessaires.

1 Suite de Collatz

La suite de Collatz de valeur initiale a est définie par $u_0 = a$ et, pour $n \in \mathbb{N}$,

$$u_{n+1} = \begin{cases} 3u_n + 1 & \text{si } u_n \text{ est impair} \\ u_n/2 & \text{si } u_n \text{ est pair} \end{cases}$$

Dans un premier temps, on emploie une boucle `for`.

Exercice VIII.1 — Valeur
Écrire une fonction `Collatz(a, n)` *qui détermine terme d'indice n de la suite de Collatz de terme initial a.*

Par exemple, pour $u_0 = 13$, la suite est $13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1, \ldots$.
La suite $13, 40, 20, 10, 5, 16, 8, 4, 2, 1, 4, 2, 1$ est le vol depuis $a = 13$.

1.1 Valeurs caractéristiques

Il semble[1] que, pour toute valeur initiale $a \geqslant 1$, la suite atteint la valeur 1 après un nombre fini d'itérations et devient alors périodique.
Dans les exercices on supposera qu'on a $a \geqslant 1$ sans avoir besoin de le vérifier.

Exercice VIII.2 — Longueur de vol
Écrire une fonction `longueurCollatz(a)` *qui détermine le premier entier n tel que $u_n = 1$ pour la suite de Collatz de terme initial a.*

On constate aussi que les valeurs de u_n peuvent être grandes, même pour u_0 assez petit. Par exemple, pour $u_0 = 703$, on a $u_{82} = 250504$.

Exercice VIII.3 — Hauteur de vol
Écrire un programme `hauteurCollatz(a)` *qui détermine l'entier maximal atteint par la suite de Collatz de terme initial a.*

1. C'est une conjecture non prouvée.

Exercice VIII.4 — Hauteur de vol, bis
Modifier la fonction précédente pour qu'elle renvoie, en plus l'indice n en lequel u_n atteint son maximum.

1.2 Seuils

Nous allons maintenant détecter les valeurs de a en lesquelles les caractéristiques ci-dessus dépassent une valeur posée.

Exercice VIII.5 — Longueur à atteindre
Écrire une fonction `longueurSeuil(s)` *qui renvoie le plus petit entier a tel que la longueur de vol de la suite de Collatz de premier terme a est supérieure à s. La fonction renverra aussi la longueur atteinte.*
`longueurSeuil(100)` *renverra* `(27, 111)`,
`longueurSeuil(250)` *renverra* `(6171, 261)`.

Exercice VIII.6 — Hauteur à atteindre
Écrire une fonction `hauteurSeuil(s)` *qui calcule le plus petit entier a tel que la hauteur de vol de la suite de Collatz de premier terme a est supérieure à s. La fonction renverra le triplet formé de la valeur initiale a, de la hauteur atteinte et de l'indice n en le quel la hauteur est atteinte.*
On recevra le résultat de `hauteurCollatz` *par l'instruction*

```
h, n = hauteurCollatz(a)
```

`hauteurSeuil(10000)` *renverra* `(13120, 255, 15)`,
`hauteurSeuil(10**9)` *renverra* `(1570824736, 77671, 71)`.

2 Exercices supplémentaires

Exercice VIII.7 — Logarithme entier
Écrire une fonction `logEntier(n)` qui renvoie l'entier p tel que $2^p \leqslant n < 2^{p+1}$.
On n'utilisera pas la fonction logarithme.

Un nombre est dit **parfait** s'il est la somme de ses diviseurs, distincts de lui-même, ses diviseurs **propres**. Les deux premiers nombres parfaits sont $6 = 1 + 2 + 3$ et $28 = 1 + 2 + 4 + 7 + 14$.

Exercice VIII.8 — Nombres parfaits
Trouver les deux suivants.
On pourra écrire une fonction qui calcule la somme des diviseurs propres d'un entier.

Charger la fonction `randint` du module `random` à l'aide de l'instruction

```
from random import randint
```

On simule un tirage de pile-ou-face par le choix aléatoire d'un entier 0 ou 1 (`randint(0,1)`).

Exercice VIII.9 — Hasard
Écrire une fonction qui fait des tirages successifs jusqu'à tirer 3 fois 1 consécutivement et renvoie le nombre de tirages qui ont été nécessaires.
Par exemple si les tirages donnent 0,1,1,0,0,1,0,1,1,0,0,0,1,0,1,1,1,...
la valeur renvoyée doit être 17.

Exercice VIII.10 — Somme de deux termes
Écrire une fonction `somme2(liste, k)` *qui renvoie deux indices i et j, $i < j$, tels que* `liste[i] + liste[j] = k`*; la fonction renverra* `(-1, -1)` *s'il n'existe pas de tels indices.*
Écrire une fonction plus rapide (une seule boucle) si on suppose que la liste est triée par ordre croissant.

3 Solutions

Solution de l'exercice VIII.1 - Penser à effectuer une division entière pour ne pas transformer les nombres en flottants.

```
def Collatz(a, n):
    u = a
    for i in range(n):
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u + 1
    return u
```

Solution de l'exercice VIII.2 -

```
def longueurCollatz(a):
    u = a
    n = 0
    while u > 1:
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u + 1
        n = n + 1
    return n
```

Solution de l'exercice VIII.3 -

```
def hauteurCollatz(a):
    u = a
    maximum = a
    while u > 1:
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u + 1
        if u > maximum:
            maximum = u
    return maximum
```

Solution de l'exercice VIII.4 -

```
def hauteurCollatz(a):
    u = a
    maximum = a
    n = 0
    n_max = 0
    while u > 1:
        n = n + 1
        if u%2 == 0:
            u = u//2
        else:
            u = 3*u + 1
        if u > maximum:
```

```
            maximum = u
            n_max = n
    return maximum, n_max
```

Solution de l'exercice VIII.5 -

```
def longueurSeuil(s):
    l= 1
    a = 1
    while l < s:
        a = a + 1
        l = longueurCollatz(a)
    return a, l
```

Solution de l'exercice VIII.6 -

```
def hauteurSeuil(s):
    h = 1
    a = 1
    n = 0
    while h < s:
        a = a + 1
        h, n = hauteurCollatz(a)
    return h, a, n
```

Solution de l'exercice VIII.7 -
On calcule les puissances de 2 jusqu'au moment où on dépasse n strictement.
On doit alors diminuer l'exposant de 1.
On calcule les puissances dans une variable pour diminuer les calculs.

```
def logEntier(n):
    """Entrée : un entier strictement positif
        Sortie : le plus grand entier p tel que 2**p <= n"""
    p = 0
    puiss2 = 1
    while puiss2 <=  n:
        p = p + 1
        puiss2 = 2*puiss2
    return p-1
```

Solution de l'exercice VIII.8 -

```
def sommeDiviseurs(n):
    somme = 0
    for i in range(1,n): # On commence par 1, on finit à n-1
        if n % i == 0:
            somme = somme + i
    return somme
```

```
k  = 29
for i  in range(2):
    while sommeDiviseurs(k) != k:
        k = k + 1
    print(k)
    k = k + 1
```

On trouve 496 et 8 128.

Solution de l'exercice VIII.9 -

```
from random import randint

def nbTirages():
    nb_tirages = 0
    nb_un = 0
    while nb_un < 3:
        k = randint(0,1)
        nb_tirages = nb_tirages + 1
        if k == 1:
            nb_un = nb_un + 1
        else:
            nb_un = 0
    return nb_tirages
```

Solution de l'exercice VIII.10 -

```
def somme2(liste, k):
    n = len(liste)
    for i in range(n-1):
        for j in range(i+1, n):
            if liste[i] + liste[j] == k:
                return (i, j)
    return (-1, -1)
```

Si la liste est triée, on part des extrémité, on calcule la somme. Si la somme est trop petite, on doit augmenter donc on incrémente le petit indice, sinon on décrémente le grand indice. On s'arrête dès que les indices ont croisé.

```
def somme2(liste, k):
    n = len(liste)
    i = 0
    j = n-1
    while i < j:
        s = liste[i] + liste[j]
        if s == k:
            return (i, j)
        elif s < k:
            i = i + 1
        else:
            j = j - 1
    return (-1, -1)
```

TP IX

Jeux sur les chiffres

1 Retournement

Dans cette partie, on étudie le retournement d'un nombre : par exemple 147 devient 741.
Pour retourner un nombre, on propose l'algorithme suivant :

1. on calcule la liste de ses chiffres, par exemple 147 donne `[1, 4, 7]`,
2. on retourne cette liste, on obtient ici, `[7, 4, 1]`,
3. on revient à l'entier associé à la liste $(7 * 10 + 4) * 10 + 1 = 741$

La clé pour le calcul des chiffres est de remarquer que le chiffre des unités de n est `n%10` et que les autres chiffres sont obtenus à partir de `n//10` ; on recommence ensuite.
Dans le cas de 741,

- le chiffre des unités est `1 = 741%10` et les autres chiffres proviennent de `74 = 741//10`
- le chiffre des dizaines est `4 = 74%10` et les chiffres suvants se calculent avec `7 = 74//10`
- le chiffre des centaines est `7 = 7%10` et il reste `0 = 7//10` donc on s'arrête.

On remarque que les chiffres sont calculés "à l'envers" ; on doit déterminer leur nombre afin de définir un tableau de taille adaptée et le remplir de gauche à droite.

Pour calculer le nombre de chiffres de n, on remarque que si n admet p chiffres alors `n//10` en admet $p-1$; on divise donc par 10 tant qu'on peut, c'est-à-dire tant que n est non nul en ajoutant 1 à chaque étape au nombre de chiffres.
On notera qu'il est cohérent de considérer que 0 s'écrit avec 0 chiffre.

Exercice IX.1 — Nombre de chiffres
Écrire une fonction `nb_chiffres(n)` *qui calcule le nombre de chiffres de n.*
On supposera que n, sans avoir besoin de le prouver, est un entier positif.

Pour calculer les chiffres de n, on a vu qu'on remplissait une liste à partir de la droite ; si p est le nombre de chiffres, on calcule d'abord le chiffre des unités que l'on place à la position $p-1$, puis on calcule le chiffre à la position $p-2$ jusqu'à 0.

Exercice IX.2 — Chiffres
Écrire une fonction `chiffres(n)` *qui calcule les chiffres de n sous forme d'une liste.*

On a besoin de convertir une liste de chiffres en un nombre.
La méthode suggéré ci-dessus est la méthode de Hörner qui minimise les calculs.
On peut l'écrire de la façon suivante.

```
def nombre(liste):
    p = len(liste)
    n = 0
    for i in range(p):
        n = 10*n + liste[i]
    return n
```

On pourra vérifier qu'on a bien `nombre(chiffres(n)) = n` pour quelques valeurs de n.
Pour retourner une liste, on pourra utiliser

```
etsil = liste[ : : -1]
```

Exercice IX.3 — Retournement d'un entier
Écrire une fonction `retournement(n)` *qui le nombre obtenu par retournement de* n.

2 Nombres de Lychrel

Un nombre est un palindrome s'il est égal à son nombre retourné.

Exercice IX.4 — Test de palindrome
Écrire une fonction `test(n)` *qui renvoie* `True` *ou* `False` *selon que* n *est un palindrome ou non.*

On définit une suite (u_n) par son premier terme, u_0, et par la relation de récurrence :
u_{n+1} est la somme de u_n et du miroir de u_n.
Par exemple si on pose $u_0 = 79$ alors $u_1 = 79 + 97 = 176$, $u_2 = 176 + 671 = 847$, $u_3 = 847 + 748 = 1595$, $u_4 = 1595 + 5951 = 7546$, $u_5 = 7546 + 6457 = 14003$ et $u_6 = 14003 + 30041 = 44044$.
On remarque que u_6 est un palindrome.

Exercice IX.5 — Calcul des termes de la suite
Écrire une fonction `suite(u0, n)` *qui calcule la valeur de* u_n *pour la suite de premier terme* u_0 *définie ci-dessus.*

Si on part de $u_0 = 28$ alors $u_1 = 28 + 82 = 110$, $u_2 = 110 + 11 = 121$ qui est aussi un palindrome.
En fait la suite parvient souvent à un nombre palindrome rapidement : les nombres qui ne semblent pas aboutir à un palindrome sont appelés nombres de **Lychrel**. Le plus petit nombre de Lychrel est 196.
On admettra que pour $a < 10000$, *an* est un nombre de Lychrel si et seulement si aucun des termes u_n pour $0 \leqslant n < 25$ n'est un palindrome.

Exercice IX.6
Combien y-a-t-il de nombres de Lychrel inférieurs à 1000 ?
On affichera ces nombres à l'écran.

Exercice IX.7
Déterminer le premier entier a tel que la suite (u_n) *de premier terme a n'aboutit à un palindrome que pour* u_{24}. *On admettra que cela advient entre 1 et 195.*
On donnera aussi la valeur de u_{24}.

3 Exercices tirés du projet Euler

Exercice IX.8 — Projet Euler 20
Que vaut la somme des chiffres de 100 ! ?
Python manie les entiers sans limitation.

On peut remarquer que 145 =!1+!4+!5. Il existe un second entier vérifiant cette propriété d'être la somme des factorielles de ses chiffres (en dehors des cas triviaux 1 =!1 et 2 =!2).

Exercice IX.9 — Projet Euler 34
Déterminer l'entier $n > 145$ *qui est la somme des factorielles de ses chiffres.*

Exercice IX.10 — Projet Euler 56
Quelle est la somme des chiffres maximale pour les entiers de la forme a^b *avec* a *et* b *entiers,* $a < 100$ *et* $b < 100$.

4 Solutions

Solution de l'exercice IX.1 -

```
def nb_chiffres(n):
    p = 0
    while n > 0:
        p = p + 1
        n = n//10
    return p
```

Solution de l'exercice IX.2 -

```
def chiffres(n):
    p = nb_chiffres(n)
    chf = [0]*p
    for i in range(p):
        c = n%10
        n = n//10
        chf[p-1-i] = c
    return chf
```

Solution de l'exercice IX.3 -

```
def retournement(n):
    liste = chiffres(n)
    etsil = liste[ : : -1]
    return nombre(etsil)
```

Solution de l'exercice IX.4 -

```
def test(n):
    m = retournement(n)
    return n == m
```

Solution de l'exercice IX.5 -

```
def suite(u0, n):
    u = u0
    for i in range(n):
        u = u + retournement(u)
    return u
```

Solution de l'exercice IX.6 -

```
nb = 0
for a in range(1, 1000):
    u = a
    n = 0
    while n < 50 and not test(u):
        u = u + retournement(u)
        n = n + 1
    if n == 50:
        print(a)
        nb = nb + 1
print(nb)
```

Il y en a 13.

Solution de l'exercice IX.7 -

```
for a in range(1, 196):
    u = a
    n = 0
    while not test(u):
        u = u + retournement(u)
        n = n + 1
    if n == 24:
        print(a)
        print(u)
        break
```

Pour $a = 89$, $u_{24} = 8813200023188$

Solution de l'exercice IX.8 -

```
def facto(n):
    f = 1
    for i in range(n):
        f = f*(i+1)
    return f

l = chiffres(facto(100))
somme = 0
for x in l:
    somme = somme + x
print("La somme des chiffres de 100! est",somme)
```

648

Solution de l'exercice IX.9 -

```
fact = [1, 1, 2, 6, 24, 120, 720, 5040, 40320, 362880]
for n in range(10000, 100000):
    l = chiffres(n)
    s = 0
    for x in l:
        s = s + fact[x]
    if n ==  s:
        print(n)
        break
```

40585

Solution de l'exercice IX.10 -

```
max = 1
a_max = 1
b_max = 1
for a in range(100):
    for b in range(101):
        s = sum_c(a**b)
        if s > max:
            print(a, b, s)
            max = s
            a_max = a
            b_max = b
print(a_max, b_max, max)
```

La somme est 972 pour 99^{95}.

TP X

LISTES DYNAMIQUES

Dans ce T.P. nous allons utiliser utiliser le caractère dynamique des listes python.
On peut construire sans connaître à l'avance la taille de la liste.
Une première conséquence est qu'il est possible de construire une liste de taille n en partant d'une liste vide et y adjoignant n éléments. On a maintenant deux possibles pour construire une liste de taille donnée, la construction avec append étant légèrement plus lente.
On pourra tester les fonctions avec la liste

```
Ex = [ 0, -2,  1, -2, -2, -1,  2,  3,  3,  0,
       0,  0,  1,  2,  0, -2, -1, -1,  1,  3,
       3,  1,  3,  0,  2, -1, -1,  1,  1,  1 ]
```

1 Exercices

Exercice X.1 — Termes positifs
Écrire une fonction `positifs(L)` *qui renvoie la liste des termes strictement positifs de* `L`*, dans le même ordre.*
`positifs(Ex)` *renvoie* `[1, 2, 3, 3, 1, 2, 1, 3, 3, 1, 3, 2, 1, 1, 1]`

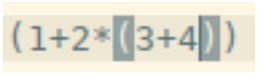
Exercice X.2 — Recherche
Écrire une fonction `positions(x, L)` *qui renvoie la liste des indices i en lesquels* `L[i]` *vaut x, la liste renvoyée sera vide si x n'apparaît pas dans la liste.*
`positions(-1, Ex)` *renvoie* `[5, 16, 17, 25, 26]`

Exercice X.3 — Positions du maximum
Écrire une fonction `indices_max(L)` *qui renvoie la liste des indices i en lesquels* `L[i]` *vaut le maximum de la liste. On pourra, dans un premier temps, utiliser l'exercice* **X.2** *après avoir calculé la valeur du maximum. Il existe cependant une méthode qui permet de trouver le résultat en ne parcourant qu'une fois la liste.*
`indices_max(Ex)` *renvoie* `[7, 8, 19, 20, 22]`

Exercice X.4 — Doublons
Écrire une fonction `sans_doublon(L)` *qui renvoie la liste des valeurs de* `L` *dans le même ordre mais sans répétitions dans des termes consécutifs.*
`sans_doublon(Ex)` *renvoie*
`[0, -2, 1, -2, -1, 2, 3, 0, 1, 2, 0, -2, -1, 1, 3, 1, 3, 0, 2, -1, 1]`

2 Complément : parenthésage

On veut étudier le parenthésage d'expressions arithmétiques ou du code source d'un programme. Il est utile de déterminer la parenthèse ouvrante associée à une parenthèse fermante, les éditeurs montrent souvent les appariements de parenthèses.



Par exemple dans $1 + 2 * (7 - (4 - 3) * ((2 - 5) + 2 * ((12/4 - 8) + 2 * 3)))$la parenthèse ouvrante après $(4 - 3)*$ est associée à l'avant dernière parenthèse fermante.
On considère une chaîne de caractère représentant l'expression et on s'intéresse uniquement aux parenthèses classiques "(" et ")". On rappelle qu'une chaîne de caractères se manipule comme si elle était une liste de caractères, avec la particularité qu'elle n'est pas modifiable.

Exercice X.5 — Test de parenthésage
Écrire une fonction `testPar(ch)` *pour qu'elle renvoie* `True` *ou* `False` *selon que la chaîne de caractères est bien parenthésée ou non.*

Pour associer une parenthèse fermante à la parenthèse ouvrante associée, on peut remarquer que celle-ci est la dernière parenthèse ouvrante non encore associée. Pour la calculer on peut appliquer l'algorithme suivant

- On crée une liste vide pour les couples de parenthèses, `par`.
- On crée une liste vide pour les indices de parenthèses ouvrantes, `ouv`.
- On lit les caractères un par un,
  - quand on voit une parenthèse ouvrante on ajoute (`ouv.append`) sa position,
  - quand on lit une parenthèse fermante à la position j, on enlève (`ouv.pop`) la dernière valeur depuis la liste `ouv`, i, ce sera bien la dernière parenthèse ouvrante non encore associée ; on ajoute alors (`par.append`) le couple (i, j) à la liste `par`.

Dans la chaîne `"1+2*(7-(4-3)*((2-5)+2*((12/4-8)+2*3)))"` les opérations seront

chaîne lue	i	ouv	par
"1"	0	[]	[]
"1+2*("	4	[4]	[]
"2*(7-("	7	[4, 7]	[]
"-(4-3)"	11	[4]	[(7, 11)]
"...)*("	13	[4, 13]	[(7, 11)]
"...*(("	14	[4, 13, 14]	[(7, 11)]
"...-5)"	18	[4, 13]	[(7, 11), (14, 18)]
"...2*("	22	[4, 13, 22]	[(7, 11), (14, 18)]
"...*(("	23	[4, 13, 22, 23]	[(7, 11), (14, 18)]
"...-8)"	30	[4, 13, 22]	[(7, 11), (14, 18), (23, 30)]
"...*3)"	35	[4, 13]	[(7, 11), (14, 18), (23, 30), (22, 35)]
"...3))"	36	[4]	[(7, 11), (14, 18), (23, 30), ... (22, 35), (13, 36)]
"...)))"	37	[]	[(7, 11), (14, 18), (23, 30), ... (22, 35), (13, 36), (4, 37)]

Exercice X.6 — Liste des parenthèses associées
Écrire une fonction `listePar(ch)` *qui reçoit une chaîne de caractères* **supposée bien parenthésée** *et qui retourne la liste des couples d'indices de parenthèses associées.*

Exercice X.7 — Test de parenthésage bis
Écrire une fonction `mauvaisePar(ch)` *pour qu'elle renvoie* `-1` *si la chaîne est bien parenthésée ou bien un indice i indiquant la position d'une parenthèse ouvrante non fermée ou la position d'une parenthèse fermante sans parenthèse ouvrante associée.*
Par exemple `listePar("(1+2))*(5-3)")` *renvoie* `5`.

3 Solutions

Solution de l'exercice X.1 -

```
def positifs(L):
    pos = []
    for x in L:
        if x > 0:
            pos.append(x)
    return pos
```

Python propose aussi la construction

```
[x for x in L if x >=0]
```

Solution de l'exercice X.2 -

```
def positions(x, L):
    n = len(L)
    pos = []
    for i in range(n):
        if L[i] == x:
            pos.append(i)
    return pos
```

```
[i for i in range(len(L)) if L[i] == x]
```

Solution de l'exercice X.3 - La méthode naturelle

```
def indices_max(L):
    maxi = L[0]
    for x in L:
        if x > maxi:
            maxi = x
    return positions(maxi, L)
```

En un seul passage

```
def indices_max(L):
    n = len(L)
    maxi = L[0]
    pos = [0]
    for i in range(1, n):
        if L[i] > maxi:
            maxi = L[i]
            pos = [i]
        elif L[i] == maxi:
            pos.append(i)
    return pos
```

Solution de l'exercice X.4 - On n'ajoute un élément que s'il est distinct de son prédécesseur, le premier terme n'a pas à être comparé.

```
def sans_doublon(L):
    n = len(L)
    L1 = [L[0]]
    for i in range(1, n):
        if L[i] != L[i-1]:
            L1.append(L[i])
    return L1
```

Solution de l'exercice X.5 -
Il suffit de compter le nombre de parenthèses ouvrantes non encore fermées.
Un manque de parenthèses ouvrantes se voit quand ce nombre est nul alors qu'on lit une parenthèse fermante, un excès de parenthèses ouvrantes se voit à la fin si le nombre est non nul.

```
def testPar(ch):
    n = len(ch)
    ouv = 0
    for i in range(n):
        car = ch[i]
        if car == "(":
            ouv = ouv + 1
        if car == ")":
            if ouv == 0:
                return False
            else:
                ouv = ouv - 1
    return ouv == 0
```

Solution de l'exercice X.6 -

```
def listePar(ch):
    n = len(ch)
    par = []
    ouv = []
    for i in range(n):
        car = ch[i]
        if car == "(":
            ouv.append(i)
        if car == ")":
            j = ouv.pop()
            par.append((j, i))
    return par
```

Solution de l'exercice X.7 -
Il suffit de compter le nombre de parenthèses ouvrantes non encore fermées.
Un manque de parenthèses ouvrantes se voit quand ce nombre est nul alors qu'on lit une parenthèse fermante, un excès de parenthèses ouvrantes se voit à la fin si le nombre est non nul.

```
def mauvaisePar(ch):
    n = len(ch)
    ouv = []
    for i in range(n):
        car = ch[i]
        if car == "(":
            ouv.append(i)
        if car == ")":
            if ouv == []:
                return i
            else:
                j = ouv.pop()
    if ouv == []:
        return -1
    else:
        return ouv.pop()
```

On peut aussi renvoyer toutes les paires de parenthèses, y comprises celles qui ne sont pas appariées, en indiquant avec `-1` la parenthèses manquante.

```
def parentheses(ch):
    n = len(ch)
    par = []
    ouv = []
    for i in range(n):
        car = ch[i]
        if car == "(":
            ouv.append(i)
        if car == ")":
            if ouv == []:
                par.append((-1, i))
            else:
                j = ouv.pop()
                par.append((j, i))
    while ouv != []:
        i = ouv.pop()
        par.append((i, -1))
    return par
```

TP XI

Arithmetique

Dans ce T.P. nous allons utiliser les outils vus jusqu'à présent pour étudier quelques constructions arithmétiques. On rappelle que k divise n si et seulement si `n%k` vaut 0 (pour $k > 0$).

1 Nombres premiers

Exercice XI.1 — Diviseurs
Écrire une fonction `diviseurs(n)` *qui renvoie la liste de tous les diviseurs de n, 1 et n compris. On supposera que n un entier strictement positif.*
`diviseurs(30)` *renvoie* `1, 2, 3, 5, 6, 10, 15, 30]`

On rappelle qu'un entier strictement positif est **premier** si et seulement si il admet exactement deux diviseurs, 1 et lui-même, ce qui exclut 1 des nombres premiers.

Exercice XI.2 — Test de primalité 1
En utilisant la fonction précédente écrire une fonction `premier(n)` *qui renvoie* `True` *ou* `False` *selon que n est premier ou non.*

On sait aussi que, si n n'est pas premier, il admet un diviseur m avec $2 \leqslant m \leqslant \sqrt{n}$.

Exercice XI.3 — Test de primalité 2
En utilisant cette propriété écrire une fonction `premier(n)` *qui renvoie* `True` *ou* `False` *selon que n est premier ou non. Cette fonction ne devra pas faire plus que* $\sqrt{n}$ *tests de divisibilité.*

Il est parfois utile de disposer de la liste des premiers nombres premiers.
Le crible d'Erathostène (vers -220 avant J.C.) permet de le faire de manière efficace. La description qui suit est celle de Wikipedia.
L'algorithme procède par élimination : il s'agit de supprimer d'une table des entiers de 2 à N tous les multiples d'un entier (autres que lui-même). En supprimant tous ces multiples, à la fin il ne restera que les entiers qui ne sont multiples d'aucun entier à part 1 et eux-mêmes, et qui sont donc les nombres premiers.

1. On commence par rayer les multiples de 2, puis les multiples de 3 restants, puis de 5, et ainsi de suite en rayant à chaque fois tous les multiples du plus petit entier restant.
2. On peut s'arrêter lorsque le carré du plus petit entier restant est supérieur au plus grand entier restant, car dans ce cas, tous les non-premiers ont déjà été rayés précédemment.
3. À la fin du processus, tous les entiers qui n'ont pas été rayés sont les nombres premiers inférieurs à N.

On pourra utiliser une liste de booléens pour implémenter la table des entiers à rayer.

Exercice XI.4 — Crible
Écrire une fonction `crible(n)` *qui renvoie la liste des entiers premiers de 1 à n en utilisant cette méthode.*
`crible(30)` *renvoie* `[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]`

2 P.G.C.D

Le **PGCD** de deux entier a et b est le plus grand entier, noté $a \wedge b$ qui divise a et b.
Par exemple $12 \wedge 30 = 6$, $25 \wedge 33 = 1$, $91 \wedge 169 = 13$.
le TP5 proposait d'écrire une fonction simple

```
def pgcd(a,b):
    commun = 1
    for k in range(1, min(a,b)+1):
        if (a % k == 0) and (b % k == 0 ):
            commun = k
    return commun
```

On peut prouver que, si $b = 0$, $a \wedge b = a$ et si $b \neq 0$ et $a = b.d + r$ est la division euclidienne de a par b (`d = a//b` et `r = a%b`) alors $a \wedge b = b \wedge r$.
C'est l'algorithme d'Euclide (vers -300 avant J.C.)

Exercice XI.5 — Diviseurs
Écrire une fonction `pgcd(a, b)` *qui calcule le PGCD de a et b en utilisant l'algorithme d'Euclide.*

Le théorème de Bezout énonce qu'il existe des entiers p et q tels que $a.p + b.q = a \wedge b$. Il n'y a pas unicité des coefficients p et q ; par exemple
$13 = 91 \wedge 169 = 2 \times 91 - 169 = -11 \times 91 + 6 \times 169 = 15 \times 91 - 8 \times 169 = \cdots$
On remarque que p ou q peut être négatif.
Pour calculer les coefficients p et q on propose l'algorithme suivant.

- Si $a^b = b$, on choisit $p = 0$ et $q = 1$.
- Si $r = a \wedge b < b$, on cherche le premier entier positif p tel que `p*a % b == r` (on admet qu'il existe) et on pose alors `q = -((p*a)//b)`.

Exercice XI.6 — Coefficients de Bezout
Écrire une fonction `bezout(a, b)` *qui calcule un couple d'entier* `p, q)` *tels que $a.p + b.q = a \wedge b$.*

3 Compléments (plus difficiles)

Exercice XI.7 — Diviseurs
Écrire une fonction `decomposition(n)` *qui calcule la liste des diviseurs premiers de n avec répétition et par ordre croissant.*
`decomposition(180)` *renvoie* `[2, 2, 3, 3, 5]`

Exercice XI.8 — Intersection
Écrire une fonction `inter(L1, L2)` *qui calcule la liste des éléments communs aux deux listes* `L1` *et* `L2` *avec répétitions. On écrira un algorithme rapide qui utilise le fait que* `L1` *et* `L2` *sont* **croissantes**.
`inter([2, 5, 5, 7, 11], [2, 3, 3, 5, 5, 5])` *renvoie* `[2, 5, 5]`

Exercice XI.9 — PGCD avec les décompositions
En déduire une fonction `pgcd_dec(a, b)` *qui calcule le PGCD en utilisant que sa décomposition est l'intersection des décompositions en facteurs premiers de a et de b.*

Exercice XI.10 — Coefficients de Bezout bis
Écrire une fonction `bezout(a, b)` *qui calcule un couple d'entier* `p, q)` *tels que $a.p + b.q = a \wedge b$. On donnera un algorithme plus rapide que précédemment en suivant l'algorithme d'Euclide.*

4 Solutions

Solution de l'exercice XI.1 -

```
def diviseurs(n):
    div = []
    for k in range(1, n+1):
        if n%k == 0:
            div.append(k)
    return div
```

On teste les divisibilités pour les entiers supérieurs à $\frac{n}{2}$, ce qui est inutile.

```
def diviseurs(n):
    div = []
    for k in range(1, n//2+1):
        if n%k == 0:
            div.append(k)
    div.append(n)
    return div
```

Solution de l'exercice XI.2 -

```
def premier(n):
    return len(diviseurs(n)) == 2
```

Solution de l'exercice XI.3 -

```
def premier(n):
    p = 2
    while p*p <= n:
        if n%p == 0:
            return False
        p = p + 1
    return p > 1 # 1 n'est pas premier
```

Solution de l'exercice XI.4 -

```
def crible(n):
    era = [True]*(n+1)
    prem = []
    era[0] = False
    era[1] = False
    for k in range(2, n+1):
        if era[k]:
            prem.append(k)
            for i in range(2*k, n+1, k):
                era[i] = False
    return prem
```

Solution de l'exercice XI.5 -

```
def pgcd(a, b):
    while b > 0:
        r = a%b
        a = b
        b = r
    return a
```

Solution de l'exercice XI.6 -

```
def bezout(a, b):
    r = pgcd(a, b)
    if r == b:
        return (0, 1)
    else:
        p = 1
        while p*a % b != r:
            p = p + 1
        return (p, -((p*a)//b))
```

Solution de l'exercice XI.7 -

```
def decomposition(n):
    facteurs = []
    d = 2
    while d <= n:
        if n%d == 0:
            facteurs.append(d)
            n = n//d
        else:
            d = d + 1
    return facteurs
```

Solution de l'exercice XI.8 -

```
def inter(L1, L2):
    n1 = len(L1)
    n2 = len(L2)
    int = []
    i1 = 0
    i2 = 0
    while i1 < n1 and i2 < n2:
        if L1[i1] == L2[i2]:
            int.append(L1[i1])
            i1 = i1 + 1
            i2 = i2 + 1
        elif L1[i1] < L2[i2]:
            i1 = i1 + 1
        else:
            i2 = i2 + 1
    return int
```

Solution de l'exercice XI.9 -

```
def produit(liste):
    p = 1
    for x in liste:
        p = p*x
    return p
```

```
def pgcd_dec(a, b):
    d1 = decomposition(a)
    d2 = decomposition(b)
    return produit(inter(d1, d2))
```

Solution de l'exercice XI.10 - On note a_k les valeurs successives de a.
On a $a_0 = a$ et $a_1 = b$, puis $a_{k+2} = a_k \% a_{k+1}$.
On a $a_0 = a = 1.a + 0.b$ et $a_1 = b = 0.a + 1.b$. On cherche p_k et q_k tels que $a_k = p_k.a + q_k.b$.
L'égalité $a_{k+2} = a_k \% a_{k+1} = a_k - (a_k // a_{k+1}) * a_{k+1}$ donne $p_{k+2} = p_k - (a_k // a_{k+1}) * p_{k+1}$ et $q_{k+2} = q_k - (a_k // a_{k+1}) * q_{k+1}$

```
def bezout(a, b):
    p_a = 1
    q_a = 0
    p_b = 0
    q_b = 1
    while b > 0:
        r = a%b
        d = a//b
        a = b
        b = r
        p = p_a - d*p_b
        q = q_a - d*q_b
        p_a = p_b
        q_a = q_b
        p_b = p
        q_b = q
    return a, p_a, q_a
```

TP XII

A.D.N.

On sait depuis le début des années 1950 que l'information génétique d'un organisme vivant est mémorisée dans la molécule d'ADN de ses chromosomes. Cette molécule, constituée de deux brins complémentaires, est un long (plusieurs milliards) enchaînement de nucléotides de quatre types différents (distingués par leur base azotée : adénine, thymine, cytosine et guanine) désignés par les lettres A, T, C et G.
Séquencer le génome d'un organisme, c'est établir l'enchaînement de ces nucléotides. Le premier génome entièrement séquencé, celui de la bactérie HAEMOPHILUS INFUENZAE en 1995, comportait moins de deux millions de nucléotides. Des génomes d'organismes évolués peuvent comporter plusieurs milliards de nucléotides. La molécule d'ADN apparaît ainsi comme un très long texte écrit avec les quatre lettres A, T, C, G.
À titre de comparaison un livre de poche de 1 000 pages comporte environ deux millions de caractères...
Enfin, seules certaines parties de cet enchaînement, appelées régions codantes ont un **sens**, c'est à dire, qu'elles sont susceptibles de coder une protéine.
L'objet de ce problème est de déterminer la position des régions codantes.

La bibliothèque `random` possède différentes fonctions.
On l'importera par `import random as rd`.

- `randint` telle que `random.randint(a, b)` renvoie un entier choisi aléatoirement parmi les entiers a, $a+1$, ..., b. Pour choisir un élément au hasard dans une liste on peut donc écrire

```
def choisir(ens):
    n = len(ens)
    k = rd.randint(0, n-1)
    return ens[k]
```

- Cette dernière fonction existe dans la bibliothèque : `rd.choice(ens)` renvoie une valeur choisie aléatoirement parmi les éléments de la liste ou chaîne de caractère (`ens`) passée en argument. Ainsi on choisit un des caractères A, C, G ou T de manière équiprobable par `rd.choice('ACGT')`

Exercice XII.1 — Génération
Écrivez une fonction `ADN` *qui reçoit un entier n pour paramètre et qui renvoie une chaîne de caractères* `chaine` *de longueur n et constituée uniquement des caractères A, C, T, G répartis aléatoirement.*

Dans toute la suite les chaînes de caractères seront supposées ne contenir que les caractères A, C, T et G. Dans les fonctions on utilisera pour les exemples la chaîne

```
ch0 = 'GATGCATTAGTAAGGGGTAGA'
```

Exercice XII.2 — Compter
Écrire une fonction `nb_lettres(chaine, lettre)` *qui reçoit une chaîne ADN et un caractère, A, C, T ou G, et renvoie le nombre d'occurrences du caractère dans la chaîne.*

`nb_lettres(ch0, 'A')` doit renvoyer 7.

Exercice XII.3 — Statistiques
Écrire une fonction `composition(chaine)` *qui renvoie les statistiques de* `chaine` *: nombre d'occurrences et pourcentage de chaque caractère.*

```
La chaîne GATGCATTAGTAAGGGGTAGA contient
 7 fois A, soit 33.33 %
 1 fois C, soit  4.76 %
 8 fois G, soit 38.10 %
 5 fois T, soit 23.81 %
```

Exercice XII.4
Écrire une fonction `renverse(chaine)` *qui prend comme argument une chaîne de caractères et la renvoie dans l'ordre inverse.*

`renverse(ch0)` renverra `'AGATGGGGAATGATTACGTAG'`.

Mutations

Une mutation est une modification de l'information génétique dans le génome d'une cellule ou d'un virus et par conséquent une modification de la séquence de l'ADN.
Les mutations sont une des causes principales de l'évolution des espèces.

Exercice XII.5
Écrire une fonction `changement(chaine, k, car)` *qui calcule et renvoie une nouvelle chaîne semblable à la chaîne initiale à l'exception du caractère à la position* `k` *(*`0 <= k < len(chaine)`*) qui est remplacé par* `car`*.*

`changement(ch0, 5, 'C')` renverra `'GATGCCTTAGTAAGGGGTAGA'`.

Exercice XII.6
Écrire une fonction `mutation(chaine)` *qui renvoie la chaîne initiale après avoir remplacé un caractère à une position choisie aléatoirement par* **un autre** *caractère choisi, lui aussi, aléatoirement.*

`mutation(ch0)` peut renvoyer `'GATGCATTAGTATGGGGTAGA'`.

La probabilité de mutation d'un nucléotide dans une période d'un million d'année est estimée à $\alpha = 1,5.10^{-2}$. On approchera le nombre de mutations d'une chaîne de longueur N dans une période de T millions d'années par la partie entière de $\alpha.N.T$.
Ces mutations peuvent intervenir plus d'une fois sur le même nucléotide : le nombre de nucléotides modifiés peut donc être inférieur au nombre de mutations.

Exercice XII.7
Écrire une fonction `mutations(chaine, T)` *qui renvoie une chaîne ayant subi les mutations de puis la chaîne initiale, correspondant à une période de T millions d'années.*

`mutations(ch0, 10)` peut renvoyer `ch1 = 'GAGGCATCAGTAAGGGGTGGA'`.

Il existe des modèles plus sophistiqués de mutation.
Les nucléotides sont décomposées en deux classes :
les purines (A et G) et les pyrimidines (C et T).
Les mutations se font plus souvent dans une classe (entre A et G ou entre C et T). Le modèle de Kimura (1980) estime que la probabilité de mutation vers l'élément de la même classe est $\frac{2}{3}$ tandis que la probabilité de mutation vers un des 2 éléments de l'autre classe est de $\frac{1}{6}$.

Exercice XII.8 — Question facultative
Écrire une fonction `mutationsK(chaine)` *qui renvoie la chaîne initiale après avoir remplacé un caractère à une position choisie aléatoirement par caractère choisi selon le modèle de Kimura.*

Exercice XII.9
Écrire une fonction `difference(ch1, ch2)` *où* `ch1` *et* `ch2` *ont même longueur qui calcule une troisième chaîne, de même longueur, contenant un espace aux positions où il n'y a pas eu mutation, et un* `X` *là où il y a une différence entre* `ch1` *et* `ch2`*. La fonction imprimera ensuite les trois chaînes en les alignant pour bien repérer les mutations et contrôler les fonctions.*

`difference(ch0, ch1)` imprimera

```
GATGCATTAGTAAGGGGTAGA
GAGGCATCAGTAAGGGGTGGA
  X     X           X
```

Recherche des séquences codantes

Dans notre modèle, nous particulariserons quatre séquences de 3 nucléotides (ou **codons**) : le codon ATG appelé START, et les codons TAA, TAG et TGA, appelés codons STOP. Nous définirons une **séquence codante** comme une portion continue de la chaîne telle que :

- elle commence par un START et se termine par un STOP,
- ce START et ce STOP sont **en phase**, c'est-à-dire qu'ils sont séparés par un nombre de nucléotides multiple de 3.

Par exemple, dans la séquence `ch0`, il y un codon START commençant en 1, et 3 codons STOP : `'TAG'` en 7 et en 17 et `'TAA'` en 10. Deux couples sont en phase et correspondent donc à des régions codantes de l'ADN.

Exercice XII.10
Écrire une fonction `reperage_codons` *qui parcourt une chaîne ADN, localise et identifie les codons* START *et* STOP *en renvoyant deux listes : une liste des indices des positions de la première lettre des codons* START *et un seconde liste des indices des positions de la première lettre de codons* STOP.

`reperage_codons(ch0)` renvoie `([1], [7, 10, 17])`.

Exercice XII.11
Écrire une fonction `seq_codantes` *qui analyse la sortie de la fonction précédente pour localiser les séquences codantes en les renvoyant sous la forme de la liste des chaînes de caractères comprises (strictement) entre un codon* START *et un codon* STOP *en phase. On devra chercher les séquences codantes dans les deux sens de la chaîne.*

`seq_codantes(ch0)` renvoie `['CAT', 'CATTAG', 'ATTACG']`

1 Solutions

Solution de l'exercice XII.1 -

```
def ADN(n):
    ch = ""
    for i in range(n):
        ch = ch + rd.choice('ACGT')
    return ch
```

Solution de l'exercice XII.2 -

```
def nb_lettres(chaine, lettre):
    n = 0
    for car in chaine:
        if car = lettre:
            n = n + 1
    return n
```

Solution de l'exercice XII.3 - On peut fixer le format des nombres utilisés dans la méthode `format` :
`:3d` écrit un entier avec 3 emplacements
`:5.2f` écrit un flottant avec 5 emplacements (y compris le point de séparation) avec 2 chiffres après la vigule.

```
def composition(chaine):
   n = len(chaine)
   print('La chaîne', chaine, 'contient')
   for lettre in 'ACGT':
       k = nb_lettres(chaine, lettre)
       print('{:2d} fois {}, soit {:5.2f} %'.format(k, lettre,
           k/n*100))
```

Solution de l'exercice XII.4 - On prend les caractères dans l'ordre et on les ajoute à gauche depuis une chaîne vide.

```
def renverse(chaine):
    eniahc = ""
    for car in chaine:
       eniahc = car + eniahc
    return eniahc
```

Solution de l'exercice XII.5 -

```
def changement(chaine, k, car):
    return chaine[ :k] + car + chaine[k+1 : ]
```

Solution de l'exercice XII.6 -

```
def mutation(chaine):
    n = len(chaine)
    k = rd.randint(0, n-1)
    car = rd.choice('ACGT')
    while car == chaine[k]:
        car = rd.choice('ACGT')
```

```
    return changement(chaine, k, car)
```

Solution de l'exercice XII.7 -

```
def mutations(chaine, T):
    alpha = 1.5e-2
    N = len(chaine)
    p = int(alpha*N*T)
    for i in range(p):
        chaine = mutation(chaine)
    return chaine
```

Solution de l'exercice XII.8 - On commence par déterminer les choix possibles à partir d'un nucléotide : l'élément de la même classe apparaît 4 fois pour avoir une probabilité de tirage de $\frac{4}{6} = \frac{2}{3}$

```
def choix(car):
    if car  == "A":
        return 'GGGGCT'
    elif car  == "C":
        return 'TTTTAG'
    elif car  == "G":
        return 'AAAACT'
    else:
        return 'CCCCAG'
```

```
def mutationK(chaine):
    n = len(chaine)
    k = rd.randint(0, n-1)
    car = rd.choice(choix(chaine[k]))
    return changement(chaine, k, car)
```

```
def mutationsK(chaine, T):
    alpha = 1.5e-2
    N = len(chaine)
    p = int(alpha*N*T)
    for i in range(p):
        chaine = mutation(chaine)
    return chaine
```

Solution de l'exercice XII.9 -

```
def difference(ch1, ch2):
    n = len(ch1)
    diff = ""
    for i in range(n):
        if ch1[i] != ch2[i]:
            diff = diff + 'X'
        else:
            diff = diff + ' '
    print(ch1)
    print(ch2)
    print(diff)
```

Solution de l'exercice XII.10 -

```
def reperage_codons(chaine):
    position_start = []
    position_stop = []
    for i in range(len(chaine)-2):
        codon = chaine[i:i+3]
        if codon =='ATG':
            position_start+=[i]
        if codon == 'TAA' or codon == 'TAG' or codon == 'TGA':
            position_stop += [i]
    return(position_start, position_stop)
```

Solution de l'exercice XII.11 -

```
def seq_codantes(chaine):
    eniahc = renverse(chaine)
    sequences = []
    for ch in [chaine, eniahc]:
        start, stop = reperage_codons(ch)
        for i in start:
            for j in stop:
                if j > i  and (j-i)%3 == 0:
                    sequences.append(ch[i+3:j])
    return sequences
```

TP XIII

Polynômes

On choisit de représenter le polynôme $P = \sum_{k=0}^{n} a_k X^k$ où les a_k sont des réels
sous la forme d'une liste $[a_0, a_1, \cdots a_n]$ d'objets de type `float`.
Il est important de noter que l'on peut avoir les derniers termes nuls : $1+X+2X^2$ peut être représenté par `[1, 1, 2]` mais aussi par `[1, 1, 2, 0]` ou par `[1, 1, 2, 0, 0, 0, 0]`.
La représentation $[a_0, a_1, \cdots a_n]$ avec $a_n \neq 0$ est la représentation **réduite** du polynôme, par exemple `[1, 1, 2]` est la représentation réduite de $1 + X + 2X^2$. On notera que la représentation réduite du polynôme nul est la liste vide `[]`.

Réduction

Exercice XIII.1 — Degré
Écrire une fonction `degre(P)` *qui renvoie le degré d'un polynôme donné en paramètre.*
Par convention, elle renverra -1 *pour le polynôme nul.*

Exercice XIII.2 — Réduction
Écrire une fonction `reduire` *qui renvoie la représentation réduite d'un polynôme.*

Dans la suite toute fonction qui renvoie un polynôme devra renvoyer sa représentation réduite.

Fonction polynôme

Exercice XIII.3 — Derivation
Écrire une fonction `derive(P)` *qui renvoie le polynôme dérivé formel (sous forme de liste aussi) d'un polynôme donné en paramètre.*

Exercice XIII.4 — Évaluation
Écrire une fonction `calculNaif(x, P)` *qui renvoie la valeur en un réel* x *d'un polynôme donné* P*.*
Cette fonction utilisera `x**k` *pour calculer* x^k*.*

Pour $P = \sum_{k=0}^{n} a_k X^k$ et x réel, on observe que

$$P(x) = \sum_{k=0}^{n} a_k x^k = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + x(a_n)) \cdots)))$$

Cette façon de représenter $P(x)$ est l'algorithme de Hörner. Elle peut aussi s'écrire

$$P(x) = a_0 + x(a_1 + x(a_2 + x(a_3 + \cdots + x(a_{n-1} + x(a_n + x.0)) \cdots)))$$

Exercice XIII.5
Écrire (sur un papier !) la forme de Hörner de $P(x) = 1 + 2x + 3x^2 + x^4 - x^5$.

Exercice XIII.6
Écrire une fonction `horner(x, P)` *qui calcule* $P(x)$ *en utilisant l'algorithme de Hörner.*

Opérations algébriques

Exercice XIII.7
Écrire une fonction qui calcule la somme de 2 polynômes.

Exercice XIII.8
Écrire une fonction `produitMonome(P,a,k)` *qui calcule le produit d'un polynôme* P *par* aX^k.

Exercice XIII.9
En déduire une fonction `produitP(P1,P2)` *qui calcule le produit de 2 polynômes.*

Complément 1 : polynômes de Tchebychev

Ce sont les polynômes définis par $T_n\big(\cos(x)\big) = \cos(nx)$.
On peut alors prouver qu'on a

1. $T_0(X) = 1$
2. $T_1(X) = X$
3. $T_n(X) = 2XT_{n-1}(X) - T_{n-2}(X)$.

Exercice XIII.10
Écrire une fonction `tchebychev(n)` *qui renvoie la liste des polynômes de Tchebychev* T_0 *à* T_n.

Complément 2 : division euclidienne

Le théorème de la division euclidienne affirme que pour deux polynômes A et B avec B non nul il existe deux polynômes Q et R tels que

$$\begin{cases} A = P.B + R \\ \deg(R) < \deg(B) \text{ ou } R = 0 \end{cases}$$

Le calcul du quotient Q et du reste R se fait par étape en déterminant le termes aX^m tel que $A' = A - B.aX^m$ ait un degré strictement inférieur à celui de A (ou est nul) et en recommençant à partir de A' tant que le degré est supérieur à celui de B.

Exercice XIII.11
Écrire une fonction `division(A, B)` *qui retourne le reste et la division du quotient de* A *par* B.

1 Solutions

Solution de l'exercice XIII.1 -

```
def degre(P):
    """Entrée : une liste qui représente un polynome
       Sortie : le degré du polynôme, -1 pour le polynome nul
          """
    deg = -1
    for k in range(len(P)):
        if P[k] != 0:
            deg = k
    return deg
```

Solution de l'exercice XIII.2 - Si d est le degré, le polynôme réduit est celui où on ne garde que les termes d'indices 0 à d.

```
def reduire(P):
    """Entrée : une liste qui représente un polynome
       Sortie : une liste qui représente le même polynôme
                et dont le dernier terme est non nul"""
    d = degre(P)
    return P[0:d+1] # On doit garder P[d]
```

Solution de l'exercice XIII.3 -

```
def derive(P):
    """Entrée : une liste qui représente un polynome
       Sortie : une liste qui représente la dérivée"""
    n = len(P)
    deri=[]
    for k in range(1, n):
        deri.append(k*P[k])
    return reduire(deri)
```

Solution de l'exercice XIII.4 -

```
def calculNaif(x,  P):
    """Entrée : un nombre
                et une liste représentant un polynome
       Sortie : la valeur de P(x)"""
    n = len(P)
    val = 0
    for k in range(n):
        val = val + P[k]*x**(k)
    return val
```

Solution de l'exercice XIII.5 -
$1 + x(2 + x(3 + x(0 + x(1 - x))))$

Solution de l'exercice XIII.6 -

```
def horner(x, P):
    """Entrée : un nombre
                et une liste représentant un polynome
       Sortie : la valeur de P(x)"""
    n = len(P)
    res = 0
    for k in range(1, n+1):
        res= x*res + P[-k]
    return res
```

Solution de l'exercice XIII.7 - Il faut faire attention aux longueurs différentes.

```
def sommeP(P1,P2):
    """Entrée : deux listes
       Sortie : une liste qui représente la somme des
                polynômes représentés par les listes"""
    n1 = len(P1)
    n2 = len(P2)
    som = []
    for i in range(min(n1, n2)):
        som.append(P1[i] + P2[i])
    if n1 < n2:
        som = som + P2[n1:n2]
    elif n2 < n1:
        som = som + P1[n2:n1]
    return reduire(som)
```

Solution de l'exercice XIII.8 -

```
def produitMonome(P, a, k):
    """Entrée : une liste P, un réel a et un entier positif k
       Sortie : une liste qui représente le produit par aX**k
                du polynôme représenté par la liste P"""
    n = len(P)
    res = []
    for i in range(k):
        res.append(0)
    for i in range(n):
        res.append(a*P[i])
    return reduire(res)
```

Solution de l'exercice XIII.9 -

```
def produitP(P1, P2):
    """Entrée : deux listes
       Sortie : une liste qui représente le produit des
                deux polynômes représentés par les listes"""
    n = len(P1)
    res = []
    for k in range(n):
        res = sommeP(res, produitMonome(P2, P1[k], k))
    return reduire(res)
```

Solution de l'exercice XIII.10 -

```
def tchebychev(n):
    """Entrée : un entier positif n
       Sortie : la liste de listes représentants les
                polynômes de Tchebychev de T0 à Tn"""
    T = [[1], [0, 1]] # On initialise avec T0 et T1
    for k in range(n-1): # Il manque n-1 polynômes
        P = produitMonome(T[-1], 2, 1)
        Q = produitMonome(T[-2], -1, 0)
        T.append(sommeP(P, Q))
    return T
```

Solution de l'exercice XIII.11 -

```
def division(A, B):
    n = degre(A)
    m = degre(B)
    q = max(0, n-m+1)
    Q = [0]*q
    r = n
    R = A
    while r >= m:
        a = R[r]/B[m]
        Q[r-m] = a
        P1 = produitMonome(B, -a, r-m)
        R = sommeP(R, P1)
        r = degre(R)
    return R, Q
```

TP XIV

Exploitation de données

1 Introduction

Le but de ce T.P. est de d'utiliser le langage SQL pour exploiter une base de données.
Un base de données simple est une ensemble de données regroupées

- par ligne, ce sont les données d'un objet (ici les communes de France continentale)
- par colonnes, ce les différents type de données (population, département, ...)

Nous allons utiliser des données fournies par l'INSEE, résultat des recensements.
Ce fichier est relativement simple mais très long : plus de 35000 lignes.
On pourrait le traiter à l'aide d'un tableur (c'est son format d'origine).
Cependant même les questions simples :

- quelles sont les villes de plus de 200 000 habitants ?
- quelle est la commune la plus étendue (`surf` en km^2) ?
- quelles sont les communes dont la population a plus que doublé depuis 1968 ?

demanderaient des manipulations qui seraient longues à effectuer.
Le format choisi pour ces T.P. est `SQL Lite` : d'autres formats sont possibles (`MySQL`, `postgreSQL`, `Oracle`, ...) mais tous partagent le cœur des fonctions `SQL`.

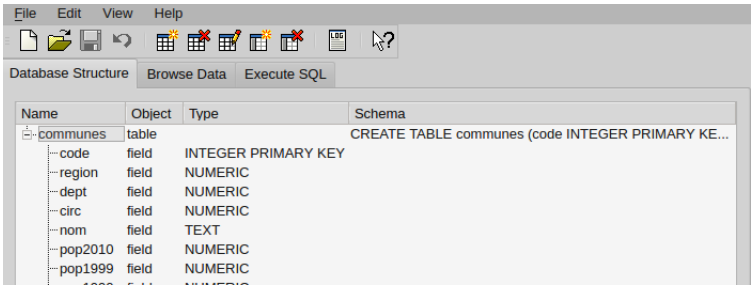
1.1 sqlitebrowser

En général l'intérêt d'une base de données est qu'elle peut être utilisée dans un réseau : la mise en place est assez lourde mais le partage d'une même base par plusieurs utilisateurs est un gros avantage. Pour ce T.P. nous allons nous contenter d'un dispositif plus simple : chaque poste exploitera sa base de données avec le logiciel SQLITEBROWSER.
Il faut commencer par copier la base de données sur le bureau (Windows) ou dans le répertoire local à votre nom (Linux). La base disponible dans le dossier de votre classe dans le disque `Public` : `communes.sqlite`.
On peut ensuite lancer le logiciel :

Windows	**Linux**
Copier le logiciel SQLITEBROWSER depuis le disque `P:` du réseau vers le bureau et le lancer	Ouvrir SQLITEBROWSER dans le menu des application, sous-menu `programmation`.

Ouvrir la base de données `communes.sqlite` depuis le menu `Files/Fichiers`,
item `Open Database/Ouvrir une base de données`.
Le logiciel présente plusieurs onglets, nous en utiliserons 3.

- L'onglet par défaut, `Database Structure/Structure de la base de données`, permet de voir l'agencement des différentes tables. Notre premier TP se contente d'une seule table. C'est ici que l'on peut s'assurer du nom des variables.
- L'onglet `Browse Data/Parcourir les données` permet de voir toutes les données comme dans un tableur : en général il n'est pas utilisable car les tables auront une grande taille.
- L'onglet `Execute SQL/Exécuter le SQL` est l'endroit où nous allons travailler.
  On entre la commande dans la fenêtre `SQL` et on la fait exécuter en cliquant sur la flèche (ou sur `[F5]`).



1.2 sqlitebrowser

L'objectif est d'extraire des informations de la base que ces informations soient déjà présentes ou qu'elles soient extrapolables. Le langage SQL est différent de Python : en effet on demande simplement le résultat souhaité sans avoir besoin d'expliquer comment y parvenir, il le fait tout seul. On parle de **langage logique**. Les questions seront appelées **requêtes**.
L'ordre des instructions dans une requête est fixé :

```
SELECT ...
FROM ...
(WHERE ...)
(GROUP BY ...)
(HAVING ...)
(ORDER BY ...)
(LIMIT ...)
(OFFSET ...)
```

Seules les deux premières instructions sont obligatoires. Les instructions à partir de `FROM` sont exécutées dans l'ordre, `SELECT` chapeaute la requête.

Select est suivi de l'énumération des informations que l'on veut voir.

From est suivi du nom de la table utilisée. Pour l'instant on se contentera d'y écrire le nom de la seule table dont on dispose : `from communes`. Le cours montrera l'utilité de cette instruction qui met en place l'aspect **relationnel** d'une base de données.

Where est suivi des conditions que doivent vérifier les informations que l'on veut visualiser.

Group by

Having Ces instructions seront l'objet du TP suivant

Order by suivi d'un nom permet de trier par ordre croissant selon les valeurs de la colonne. On peut faire suivre par `desc` si on veut un ordre décroissant.

Limit n'est utile qu'avec une instruction `Order by`. Il permet de limiter le nombre de résultats affichés

Offset n'est utile qu'avec une instruction `Limit`. Il permet de décaler les instructions. `Limit 3 Offset 4` affiche les résultats des rangs 5 à 7.

2 Requêtes

2.1 SELECT

Dans cette partie, on illustre quelques possibilités de choix de colonnes mais, pour ne pas obtenir des résultats trop longs, on pourra ajouter une instruction de filtrage, par exemple,

```
select
from
where nom = "Sainte Colombe"
```

Comme le nom d'une ville est une chaîne de caractères, il faut l'encadrer par des guillemets.
14 communes de France se nomment Sainte Colombe.

Pour faire afficher toutes les colonnes on peut écrire `select *`.
Par exemple tous les renseignements concernant les villes Sainte Colombe sont donnés par

```
select *
from communes
where nom = "Sainte Colombe"
```

Exercice XIV.1
Trouver tous les renseignement d'une ville que vous choisissez.

Exercice XIV.2
Déterminer les départements (`dept`*) et les populations (*`pop2010`*) des villes se nommant Sainte Colombe.*

Les résultats montrés peuvent contenir des calculs à partir des colonnes déjà existantes ; il est **très recommandé** de nommer le résultat du calcul avec le mot-clé **AS**.

Exercice XIV.3
Déterminer les accroissements de populations entre 1968 et 2010 (`pop2010 - pop1968 AS acc`*) des villes se nommant Sainte Colombe.*

Si on effectue une division entre des entiers on a la division euclidienne, à quotient entier. Pour obtenir la division réelle classique on doit imposer qu'un nombre soit réel. On peut, par exemple, écrire `(a + 0.0)/b`. Les valeurs affichées peuvent présenter un nombre déterminé de décimales avec `round(x,k)` qui affiche x avec k décimales.

Exercice XIV.4
Déterminer les densités de populations (`surf` *pour la superficie) des villes se nommant Sainte Colombe.*

2.2 Extraction de résultats

En combinant `ORDER BY` et `LIMIT` on peut extraire des résultats.

Exercice XIV.5
Quelles sont les 10 villes les moins peuplées de France ? On donnera aussi le département.

Exercice XIV.6
Quelles sont les 10 villes les plus densément peuplées de France ? On donnera aussi le département.

Exercice XIV.7
Quelle commune a le nom le plus long (45 signes) ? Le nom le plus court (1 lettre) ? On donnera aussi le département.
La longueur d'une chaîne de caractères s'obtient par `length(ch)`.

2.3 Where

Les conditions de sélection seront souvent basées sur des comparaisons :
les opérateurs sont `=`, `!=` (ou `<>`), `<`, `>`, `<=`, `>=`.

```
select nom, POP2010
from communes
where POP2010 > 200000
```

Exercice XIV.8
Trouver les noms des communes de moins de 10 habitants (< 10) avec leur département et leur population. Il y en a 22

Les recherches peuvent être composées par "ou" (`OR`) ou "et" (`AND`).

Exercice XIV.9
Trouver les noms des communes de plus de 30000 habitants et de surface inférieure à 5 km^2 avec leur département, leur population et leur surface.
(19, toutes en région parisienne)

Exercice XIV.10
Trouver les noms des communes dont la densité de population est supérieure à 20000 habitants par km^2 avec leur département, leur population et leur surface.
(7 communes, toutes dans la région parisienne)

Exercice XIV.11
Trouver les noms des communes dont la densité de population est inférieure à 1 habitant par km^2 avec leur département, leur population, leur surface et leur densité. (23 communes)

Exercice XIV.12
Trouver les 10 communes de plus de 100.000 habitants dont le taux d'augmentation de population a été le plus important entre 1999 et 2010.

2.4 Statistiques

On peut compter le nombre d'éléments avec `count()` ; il ne peut pas y avoir d'autres éléments affichés car il n'y a qu'une valeur à afficher pour toutes les données.

Exercice XIV.13
Combien y-a-t-il de communes dans la base ? (36205)

D'autres fonctions sont disponibles, elles calculent un résultat en fonction d'une valeur pour chaque ligne. On pourra utiliser la moyenne d'une donnée (`AVG`), le maximum (`MAX`), le minimum (`MIN`), la somme (`SUM`).

Exercice XIV.14
Quelle est la moyenne de population des communes du Nord ? (3964.2...)

Exercice XIV.15
Quelle est la somme des superficies des communes du Nord ? (5734 km^2)

Exercice XIV.16
Quelle est la différence entre le nombre de femmes et le nombre d'hommes recensés ?
1 971 799 femmes de plus !

3 En plus

Si les valeurs sont des chaînes de caractères on peut filtrer sur une partie du nom :

1. `_` représente un caractère quelconque,
2. `%` représente une suite (éventuellement vide) de caractères.
3. L'opérateur s'écrit alors `like` et non =.

Par exemple les villes dont le nom contient *truc* sont recherchées par `nom LIKE "%truc%"`

Exercice XIV.17
Trouver les communes dont le nom contient `kerque` *et leur département ; il y en a 10.*

Exercice XIV.18
La base n'est pas cohérente : la population totale (`pop2010`*) n'est pas toujours égale à la somme des populations par genre. Déterminer les communes dans lesquelles il y a une erreur. Il y en a 373.*

Un résultat statistique peut servir comme valeur dans un calcul ou une comparaison. Par exemple les villes qui ont une population égale à la population moyenne avec un écart de moins de 1 sont obtenues par la requête suivante (`abs` est la valeur absolue).

```
select dept, nom, pop2010
from communes
where abs(pop2010 - (select avg(pop2010) from communes)) < 1
```

Exercice XIV.19
Quelles sont les villes du Nord dont la population est supérieure à 10 fois la moyenne de population des villes du Nord ? (Il y en a 8)

4 Solutions

Solution de l'exercice XIV.1 -

```
select *
from communes
where nom = "Village Neuf"
```

Solution de l'exercice XIV.2 -

```
select nom, dept, pop2010
from communes
where nom = "Sainte Colombe"
```

Solution de l'exercice XIV.3 -

```
select nom, pop2010 - pop1968 AS acc
from communes
where nom = "Sainte Colombe"
```

Solution de l'exercice XIV.4 -

```
select nom, round((pop2010+0.0)/surf, 2) as densité
from communes
where nom = "Sainte Colombe"
```

Solution de l'exercice XIV.5 -

```
select nom, dept, pop2010
from communes
order by pop2010
limit 10
```

Solution de l'exercice XIV.6 -

```
select nom, round((pop2010+0.0)/surf, 2) as densité, dept
from communes
order by densité desc
limit 10
```

Solution de l'exercice XIV.7 -

```
select dept, nom, length(nom)
from communes

order by 3 desc
limit 1
```

Solution de l'exercice XIV.8 -

```
select nom, dept, POP2010
from communes
where POP2010 < 10
```

Solution de l'exercice XIV.9 -

```
select nom, dept, POP2010, surf
from communes
where POP2010 > 30000 and surf <= 5
```

Solution de l'exercice XIV.10 -

```
select nom, dept, POP2010, surf
from communes
where POP2010/surf > 20000
```

Solution de l'exercice XIV.11 -

```
select nom, dept, POP2010, surf, round(POP2010/(0.0+surf),2)
   as densite
from communes
where densite < 1
```

Solution de l'exercice XIV.12 -

```
select nom, dept, POP2010, POP1999, (POP2010 - POP1999)*100.0/
   POP1999 as taux
from communes
where POP2010 > 30000
order by taux desc
limit 10
```

Solution de l'exercice XIV.13 -

```
select count()
from communes
```

Solution de l'exercice XIV.14 -

```
select avg(pop2010)
from communes
where dept = 59
```

Solution de l'exercice XIV.15 -

```
select sum(surf)
from communes
where dept = 59
```

Solution de l'exercice XIV.16 -

```
select sum(femmes)-sum(hommes)
from communes
```

Solution de l'exercice XIV.17 -

```
select nom, dept
from communes
where nom like "%kerque%"
```

Solution de l'exercice XIV.18 -

```
select nom, femmes+hommes-pop2010 as erreur
from communes
where erreur != 0
```

Solution de l'exercice XIV.19 -

```
select nom, pop2010
from communes
where dept = 59 and pop2010 > 10*(select avg(pop2010) from
    communes where dept = 59)
```

TP XV

Exploitation statistique de données

Le but de ce T.P. est de d'utiliser les instructions d'agrégation du langage SQL lors de l'exploitation d'une base de données.

1 Présentation des regroupements

On prend l'exemple d'une base de notes de colles, courte pour les besoins de la présentation.

Colleur	Etudiant	Note
Lesec	Riri	8
Legaga	Riri	18
Lemoyen	Loulou	15
Legaga	Fifi	17
Legaga	Loulou	20
Lesec	Fifi	13
Lemoyen	Riri	15
Lemoyen	Fifi	15
Lesec	Loulou	15

Si on veut connaître par exemple les moyennes de chaque étudiant on doit regrouper les lignes par nom d'étudiant.
L'instruction **GROUP BY Etudiant** effectue ce travail virtuellement.

Colleur	Étudiant	Note
Legaga	Fifi	17
Lesec	Fifi	13
Lemoyen	Fifi	15
Lesec	Loulou	15
Lemoyen	Loulou	15
Legaga	Loulou	20
Lesec	Riri	8
Legaga	Riri	18
Lemoyen	Riri	15

- On peut alors demander les moyennes ; elles seront calculées pour chaque partie obtenue par le regroupement.

```
select AVG(Note)
from colles
group by Etudiant
```

AVG(Note)
15
16.666666667
13.666666667

- On voit qu'il manque l'information de l'étudiant : **quand on fait un regroupement, on doit toujours faire afficher la colonne qui a servi de critère de regroupement.** Il est recommandé de donner un nom au calcul

```
select Etudiant, AVG(Note) as moy
from colles
group by Etudiant
```

Etudiant	moy
Fifi	15
Loulou	16.666666667
Riri	13.666666667

- Il est par contre totalment inutile d'afficher d'autres colonnes que la colonne de critère et les valeur calculées, le logiciel donnera une réponse mais celle-ci n'a aucun sens.

```
select Etudiant, AVG(Note) as moy, Colleur
from colles
group by Etudiant
```

Etudiant	my	Colleur
Fifi	15	Legaga
Loulou	16.666666667	Lesec
Riri	13.666666667	Lesec

- Si on veut sélectionner selon des valeurs calculées, on ne peut pas le faire avec `where` car le critère de cet instruction a été utilisé **avant** le regroupement. L'instruction **HAVING**, placée après `group by` permet d'appliquer le critère. On voit ici l'utilité de nommer le calcul.

```
select Etudiant, AVG(Note) as moy
from colles
group by Etudiant
having moy > 14
```

Etudiant	moy
Fifi	15
Loulou	16.666666667

2 Requêtes

2.1 Regroupements

Exercice XV.1
Calculer le nombre de communes par département.
Quels sont les 3 départements qui comportent le plus grand nombre de communes ? (62, 2, 80)

Exercice XV.2
Quels sont les 3 départements les plus peuplés ? (59, 75, 13)

Exercice XV.3
Déterminer la densité de population par département.

Exercice XV.4
Quels est le nombre de communes par arrondissement ?

Exercice XV.5
Quels est le nom de commune le plus courant ?
14 communes s'appellent Sainte Colombe.

Exercice XV.6
Quels est le nombre de départements par région ?
En 2010, il y avait 21 régions dont la table ne donne que le numéro.
Pour compter les départements sans doublon `COUNT(dept)` *ne suffit pas, il faut écrire* `COUNT(DISTINCT dept)`.

2.2 Utilisation de `HAVING`

Exercice XV.7
Quels sont les département de population supérieure à 1.500.000 ? (59, 75, 13, 69, 92, 93).

Exercice XV.8
Quels sont les départements dont la population de la ville la plus peuplée est au moins un tiers de la population totale ? (13, 31, 75, 87, 90).

Exercice XV.9
Quels sont les arrondissements qui ne contiennent qu'une commune ? On donnera aussi le département.

Exercice XV.10
Quels sont les départements dont la population de plus de 60 ans représente plus de 15% de la population totale ? (10 départements).

Exercice XV.11
Quels sont les départements dont la population a crû de plus de 15% entre 1999 et 2010 (il y en a 8) ?

3 Requêtes imbriquées

Si le résultat d'une requête est un tableau on peut l'utiliser pour une autre requête.
Par exemple après avoir calculé la population par arrondissement,

```
select dept, arr, sum(POP2010 )as s
from communes
group by arr
```

on peut calculer la moyenne des habitants par arrondissement pour chaque département.

```
select dept, avg(s) as moyenne
from (select dept, arr, sum(POP2010 )as s
      from communes group by arr)
group by dept
order by moyenne desc
```

Si une requête produit un nombre (`count`, `avg`, `max`, ...) on peut l'utiliser dans un test ou un calcul.

Exercice XV.12
Après avoir calculé la population par département, déterminer la moyenne de la population par département (664 420,7).

Exercice XV.13
Déterminer les départements qui ont une population inférieure au quart de la moyenne de la population par département (48, 23, 5, 90, 15, 9, 4).

Exercice XV.14
Déterminer le nombre d'arrondissements par département.

Exercice XV.15
Déterminer le nombre de département qui ont 1, 2, 3, ..., 9 arrondissements.

Exercice XV.16
Quels sont les département dans lesquels sont situés les communes ayant le nom le plus courant (question XV.6) ?.
On ne devra pas écrire "Sainte Colombe".

Exercice XV.17
Déterminer le département dont la surface est la plus proche de la surface moyenne.

On pourra trier selon la valeur absolue de la différence de la surface avec la moyenne et ne garder qu'un résultat (limit 1). Le département est 15.

On veut classer les villes par leur taille en les regroupant sous la forme 1 à 9 habitants, 10 à 99 habitants, etc Pour cela la fonction `length(n)`, qui renvoie le nombre de chiffres de n, est utile.

Exercice XV.18
Déterminer le nombre de personnes habitant dans une ville comportant 1 à 9 habitants, 10 à 99 habitants, ...

3.1 Hors-programme : utilisation de `IN`

On peut aussi utiliser une table calculée dans une condition `where` : si on produit un ensemble de valeurs on peut tester l'appartenance d'une valeur (ou d'un tuple de valeurs) à cette table avec le test `x IN table`.

Exercice XV.19
Donner le nom de la ville la plus peuplée par département.

4 Solutions

Solution de l'exercice XV.1 -

```
select dept, count() as nombreCommunes
from communes
group by dept
order by 2 desc
limit 3
```

Solution de l'exercice XV.2 -

```
select dept, sum(POP2010) as pop
from communes
group by dept
order by 2 desc
limit 3
```

Solution de l'exercice XV.3 -

```
select dept, sum(POP2010)/sum(surf) as densité
from communes
group by dept
```

Solution de l'exercice XV.4 -

```
select arr, count()
from communes
group by arr
```

Solution de l'exercice XV.5 -

```
select nom, count() as nombre
from communes
group by nom
order by 2 desc
limit 1
```

Solution de l'exercice XV.6 -

```
select region, count(distinct dept) as nb
from communes
group by region
```

Solution de l'exercice XV.7 -

```
select dept, sum(POP2010) as population
from communes
group by dept
having population > 1 500 000
order by 2 desc
```

Solution de l'exercice XV.8 -

```
select dept, max(pop2010) as popMax, sum(pop2010) as pop
from communes
group by dept
having pop < 3*popMax
```

Solution de l'exercice XV.9 -

```
select dept, arr, count() as nb
from communes
group by arr
having nb = 1
```

Solution de l'exercice XV.10 -

```
select dept, 100*(sum(age60_74)+sum(age75+0.0))/sum(pop2010)
   as taux_vieux
from communes
group by dept
having taux_vieux>30
```

Solution de l'exercice XV.11 -

```
select dept, (sum(pop2010) - sum(pop1999) + 0.0)/sum(pop1999)
   as taux
from communes
group by dept
having taux > 0.15
```

Solution de l'exercice XV.12 -

```
select  dept, sum(POP2010 )as population
from communes
group by dept
```

```
select round(avg(population),1)
from (select  dept, sum(POP2010 ) as population
      from communes
      group by dept)
```

Solution de l'exercice XV.13 -

```
select dept, population
from (select  dept, sum(POP2010 ) as population
      from communes
      group by dept)
where population <  (select  avg(pop)
                     from (select  dept, sum(POP2010 ) as pop
                           from communes
                           group by dept))/4.0
order by 2 desc
```

Solution de l'exercice XV.14 -

```
select dept, count() as nbArr
from (select dept, arr,sum(pop2010) as popArr
      from communes
      group by arr)
group by dept
```

Solution de l'exercice XV.15 -

```
select nbArr, count() as nb
from (select dept, count() as nbArr
      from (select dept, arr,sum(pop2010) as popArr
            from communes
            group by arr)
      group by dept)
group by nbArr
```

Solution de l'exercice XV.16 -

```
select dept,  nom
from communes
where nom =(select nom
            from (select nom, count() as nombre
                  from communes
                  group by nom
                  order by 2 desc
                  limit 1
                 )
           )
```

Solution de l'exercice XV.17 -

```
select dept, surface
from (select dept, sum(surf) as surface
      from communes group by dept)
order by abs(surface - (select avg(surface)
                        from (select dept, sum(surf) as
                            surface
                               from communes group by dept
                              )
                       )
            )
limit 1
```

Solution de l'exercice XV.18 -

```
select sum(POP2010) as population, count() as nombre,  rang
from (select nom, dept, POP2010, length(POP2010) as rang
      from communes)
group by rang
```

Solution de l'exercice XV.19 -

```
select dept, nom, pop2010
from communes
where (dept, pop2010) in (select dept, max(pop2010)
                          from communes
                          group by dept)
```

TP XVI

Modélisation d'un amortisseur

1 Présentation

Le laboratoire de sciences industrielles dispose du matériel qui permet d'étudier une suspension de moto avec la possibilité d'effectuer des mesures.

Lors de la manipulation un capteur (un accéléromètre) calcule les accélérations d'un point de la partie mobile et enregistre celles-ci dans un fichier texte.
On souhaite traiter ce fichier afin de corréler l'expérience avec un modèle de connaissance du type oscillant amorti.

Il s'agit donc d'étudier le mouvement vertical oscillant amorti d'un solide en translation verticale dans un champ gravitationnel vertical dirigé vers le bas.
Le solide est guidé par une liaison de direction verticale possédant :

- une raideur qui crée une force, en $[N]$, proportionnelle au déplacement vertical relatif par rapport à la position d'équilibre en $[m]$ et qui s'oppose au déplacement,
- un amortissement visqueux qui crée une force verticale en $[N]$ qui s'oppose à la vitesse de déplacement relative entre le solide et la référence en $[m \cdot s^{-1}]$.

1.1 Création du fichier

Le fichier est créé selon le protocole suivant.

- On part de la position d'équilibre statique du système.
- On déplace manuellement vers le bas de $100mm$ le châssis de la moto.
- On lance une acquisition pour $6s$ avec une fréquence d'échantillonnage de $200Hz$.
- On lâche le châssis sans condition initiale de vitesse.
- On sauvegarde le fichier relatif à l'expérience sous le nom `moto1.txt` en demandant une indexation des points.
- Le fichier `moto.txt` est une suite de caractères générée par le logiciel d'acquisition.
- Il comporte 1200 lignes car l'acquisition se fait à $200Hz$ pour une acquisition de 6 secondes.
- Chaque ligne est l'enregistrement de 4 nombres :

```
   1  +0.0000000  +7.3876953  +1.9042969
   2  -0.0048828  +7.3925781  +1.8994141
   3  -0.0097656  +7.3974609  +1.8994141
 ...         ...         ...         ...
1200  -0.0048828  +7.3925781  +1.5283203
```

- La première colonne donne le numéro de la mesure, ici de 1 à 1200.
- Les trois dernières colonnes correspondent aux mesures de l'accéléromètre, ce sont des tensions qui dépendent linéairement des composantes de l'accélération dans une base orthonormée : à l'équilibre, elles mesurent l'accélération de la pesanteur. La composante verticale est écrite dans la deuxième coordonnée, c'est-à-dire la troisième colonne. C'est cette mesure que nous étudierons.
- Dans le fichier, chaque ligne est donc composée de quatre éléments.
  Ces éléments sont séparés par des caractères de tabulation, `"\t"`.
  Chaque ligne est terminée par un caractère de changement de ligne `"\n"`.

On peut ouvrir le fichier dans un éditeur pour en lire la structure.
Dans un éditeur classique les tabulations et les retours à la lignes seront traduits et on obtient un alignement vertical comme dans l'exemple ci-dessus.

1.2 Objectifs

1. Lire le fichier texte `moto1.txt`.
2. Convertir le fichier lu en une structure de données utilisable par python.
3. Transformer ces données pour obtenir une représentation de la vitesse et de la position de l'accéléromètre.
4. Rechercher les grandeurs caractéristiques associées à un modèle mécanique permettant son identification.

2 Lecture du fichier

Pour utiliser un fichier on doit ouvrir une connexion avec celui-ci par l'instruction `open`. Celle-ci a deux arguments : le nom du fichier (une chaîne de caractères) et le type qui peut être `"r"` pour la lecture, `"w"` pour l'écriture d'un nouveau fichier et `"a"` pour l'ajout à un fichier existant.
On peut importer le fichier

- comme une grande chaîne de caractères : `read`
- comme une liste de lignes : `readlines`
- ligne à ligne : `readline`

Une ligne est une portion du fichier délimitée par le caractère de retour à la ligne `"\n"`, ce caractère est conservé en fin de ligne.
On obtient donc la liste des lignes du fichier d'acquisition avec

```
## Importations
nom = 'moto.txt'
fichier = open(nom, 'r')
listeBrute = fichier.readlines()
fichier.close()
```

On rappelle l'importance de fermer la communication : `fichier.close()`.

Le fichier est lu par python dans le répertoire de travail.
Python contient un module, `os`, qui permet d'interagir avec le système de fichiers.

```
import os
```

On peut connaître le répertoire de travail avec `os.getcwd()`.
On peut se déplacer dans l'arborescence du disque avec la fonction `os.chdir` ; la méthode la plus simple pour l'utiliser est de donner le nom complet du répertoire comme argument. Cela change le répertoire de travail.

```
## Importations
import os

os.chdir("/home/profs/detrez.eric/Travail/2018-2019") # Linux
# os.chdir("H://Travail//2018-2019") # Sous Windows
```

On notera le dédoublement du séparateur `"//"` sous Windows.

On peut aussi donner le chemin complet d'accès au fichier comme paramètre.

Exercice XVI.1
Importer, comme indiqu" ci-dessus, les lignes du fichiers `moto.txt`.

On peut voir la structure non traduite des lignes dans la console ; la fonction `print` interprète les caractères spéciaux donc produit un affichage différent.

```
>>> listeBrute[55]
'         56\t-0.0097656\t+7.3974609\t+1.8945313\n'

>>> print(listeBrute[55])
         56   -0.0097656   +7.3974609   +1.8945313
```

3 Traitement initial

3.1 Conversion en nombres

On veut maintenant convertir les chaînes de caractères obtenues en données numériques.
On pourra utiliser

- une extraction `ch[:-1]` pour enlever le dernier caractère (`"\n"`)
- la méthode `split(c)` qui transforme une chaîne de caractères en une liste de chaînes de caractères qui sont les morceaux découpé par le paramètre. Par exemple

```
>>> ch = "anticonstitutionnellement"
>>> ch.split('t')
['an', 'icons', 'i', 'u', 'ionnellemen', '']
>>> ch.split('ti')
['an', 'cons', 'tu', 'onnellement']
>>> ch.split('n')
['a', 'tico', 'stitutio', '', 'elleme', 't']
```

- la fonction de conversion de type, ici `float`, pour transformer une chaîne en flottant.

Exercice XVI.2
Écrire une fonction `traduction(ch)` *qui transforme une chaîne de caractères semblable aux lignes du fichiers en une liste de nombre.*

Exercice XVI.3
Écrire les instructions (dans la partie principale) qui permettent de traduire `listeBrute` *en une liste* `liste` *dont les éléments sont des listes de 4 nombres.*

Le premier élément de la liste devra donc être `[1.0, 0.0000000, 7.3876953, 1.9042969]`.

3.2 Extraction des colonnes

Exercice XVI.4
Écrire une fonction `extraire(k,liste)` *où* k *est un entier et* `liste` *est une liste dont les éléments sont des listes de longueur* k *au moins et qui renvoie la liste des* k*-ièmes composantes.*
`extraire(2, [[1, 2, 3], [4, 5, 6]])` *doit renvoyer* `[2, 5]`

N.B. La k-ième composante correspond à l'indice $k-1$.

On peut maintenant visualiser les différentes colonnes.
On utilise pour cela le module `matplotlib`

```
import matplotlib.pyplot as plt
```

Le nom raccourci `plt` est usuel.
On affiche une liste de nombre, `X`, par

```
plt.plot(X)
plt.show()
```

Exercice XVI.5
Afficher les valeurs des 4 colonnes ; commentez.

4 Traitement des listes

Dans cette section on se propose
- de déterminer l'origine temporelle des variations,
- de convertir les résultats signifiants en unité connues,
- de calculer puis retirer l'accélération gravitationnelle,
- de calculer les vitesses puis les déplacement à partir de l'accélération.

On définit, après les instructions de la **question XVI.3**, les listes particulières :

```
tempsBrut = extraire(1, liste)
accBrute = extraire(3, liste)
```

4.1 Origine et échelle du temps

Le tracé de `accBrute` montre que l'accélération n'a pas varié immédiatement, il s'est passé du temps entre le déclenchement de la lecture des données et le lâcher de la roue. Pour déterminer l'origine du mouvement on va déterminer l'indice en lequel l'accélération a comment à varier au-delà des variations aléatoires.

Exercice XVI.6
Écrire une fonction `rechZero(liste, ecart)` *qui recherche le premier indice pour lequel la valeur de la liste s'écarte d'au moins* `ecart` *de la valeur initiale. On veut donc le premier indice i tel que* `abs(liste[i] - liste[0] >= ecart`.

On choisit dans la suite un écart assez petit [1] pour déterminer l'origine temporelle. On ajoutera, par exemple, l'instruction suivante dans la partie principale.

```
i0 = rechZero(accBrute, 1e-2) - 1
```

On a retranché 1 car le premier écart correspond à la deuxième mesure du mouvement. Avec la fréquence d'échantillonnage, on connaît le temps entre deux mesures : $\Delta t = \frac{1}{f} = 0.005\,s$.
On notera `Delta_t` une variable qui prend la valeur Δt dans la suite.

Exercice XVI.7
Écrire les instructions (dans la partie principale) qui calculent la liste des temps en secondes, notée `temps`*, en commençant avec $t = 0$ pour l'origine du phénomène (le point d'indice* `i0`*).*
La liste sera plus courte que la liste originale.

4.2 Échelle des accélérations

Lorsqu'il est au repos, c'est-à-dire pour les indices de 0 à i_0, le dispositif ne subit que l'accélération de la pesanteur. On considère donc la moyenne des valeurs de `accBrute` pour ces indices comme valeur de la mesure de cette accélération de la pesanteur. On rappelle que cette mesure est donnée en Volts, on devra la convertir en $m \cdot s^{-2}$.

Exercice XVI.8
Écrire une fonction `moy(liste)` *qui calcule la valeur moyenne de la liste.*
En déduire une valeur, V_g, de la mesure de g fournie par l'accéléromètre (dans la partie principale).

L'accéléromètre mesure l'accélération a (sous la forme d'une tension V) dans le référentiel terrestre. On a mesuré la tension $V_0 = 4,8291V$ en l'absence de toute accélération (même gravitationnelle) et on admet que la mesure de l'accélération en $m \cdot s^{-2}$, a, s'obtient à partir de la tension V de l'accéléromètre par une formule $a = \alpha V + \beta$.

1. Mais pas trop petit pour éviter les variations aléatoires.

Exercice XVI.9
Après avoit définie les variables `g = 9.81` *et* `V0 = 4.8291`*, écrire, dans la partie principale, les instructions permettant de calculer les variables* `alpha` *et* `beta` *prenant les valeurs des coefficients de la formule ci-dessus.*

L'accélération du dispositif isolé est l'accélération mesurée à laquelle on retire g.

Exercice XVI.10
Écrire les instructions (dans la partie principale) qui calculent la liste des accélérations du dispositif en m/s^2, notée `acc`*, en commençant avec $t = 0$ c'est-à-dire à partir de l'indice i_0.*
`acc` *sera donc de même longueur que la liste* `temps`*.*

On a maintenant une liste dont les valeurs sont les temps, régulièrement espacés, en lesquels le système a mesuré une accélération. Si on note t_i la valeur de `liste[i]` alors la valeur `acc[i]` est la valeur de l'accélération au temps t_i.
On peut alors représenter cette fonction accélération en fonction du temps.

```
plt.plot(temps,acc)
plt.show()
```

`matplotlib` permet d'ajouter beaucoup de personnalisations aux graphes.

```
plt.title("Accélération") # Ajout d'un titre
plt.xlabel("Temps en secondes") # Légende des abscisses
plt.ylabel("Accélération en m/s^2") # Légende des ordonnées

plt.grid(True) # Ajout d'une grille
plt.plot(temps,acc,
         color="magenta", # couleur de la ligne
         linewidth=2.5,   # épaisseur de la ligne
         linestyle="-",   # type de ligne
         label="Accélération") # nom de la fonction tracée
plt.legend() # écriture du "label"
plt.show()
```

4.3 Vitesses et positions

On cherche ensuite à évaluer les vitesses et les positions aux temps t_i.
Si on connaît une fonction en des points t_0, t_1, …, t_p et si t_i et t_{i+1} sont suffisamment proches on peut approcher la variation de ses primitives (l'intégrale) par la formule du rectangle : $F(t_{i+1}) - F(t_i) \sim t_{i+1} - t_i.f(t_i)$.
On peut alors calculer les valeurs de F de proche en proche à partir d'une valeur initiale $F(t_0)$.
Ici $t_{i+1} - t_i$ est constante, c'est Δt

Exercice XVI.11
Écrire une fonction `primitive(listeFn, dt, y0)` *où* `listeFn` *est une liste représentant les valeurs de $f(t_i)$,* `dt` *est le pas de temps et* `valeurInit` *est la valeur initiale de la primitive, $F(t_0)$ et qui renvoie une liste, de même longueur que* `listeDer`*, contenant des valeurs approchées de $F(t_i)$.*

On remarque qu'on n'utilise pas la dernière valeur de `listeFonction`.

Exercice XVI.12
Calculer et tracer le graphe des vitesses et des positions.
La vitesse initiale est 0 $m.s^{-1}$, la position initiale est -0,10 m.

On peut voir que la position ne reste pas stable à la valeur 0 à la fin des oscillations. Si on prend pour V_g la moyenne des valeurs de `accbrute` sur toute la suite, le résultat sera meilleur.

Exercice XVI.13
Enregistrer la suite des valeurs des temps et des positions dans un fichier `moto1.txt`*.*

5 Compléments : identification des caractéristiques

La réponse du système est identifiable à la solution de l'équation différentielle suivante :

$$\frac{1}{\omega_0^2}x''(t) + \frac{2\xi}{\omega_0}x'(t) + x(t) = 0$$

- où $x(t)$ est la position verticale repérée
- ξ est le coefficient d'amortissement,
- ω_0 est la pulsation propre du système

La solution telle que $x(0) = -e_0 = -0.1$ et $x'(0) = 0$ est

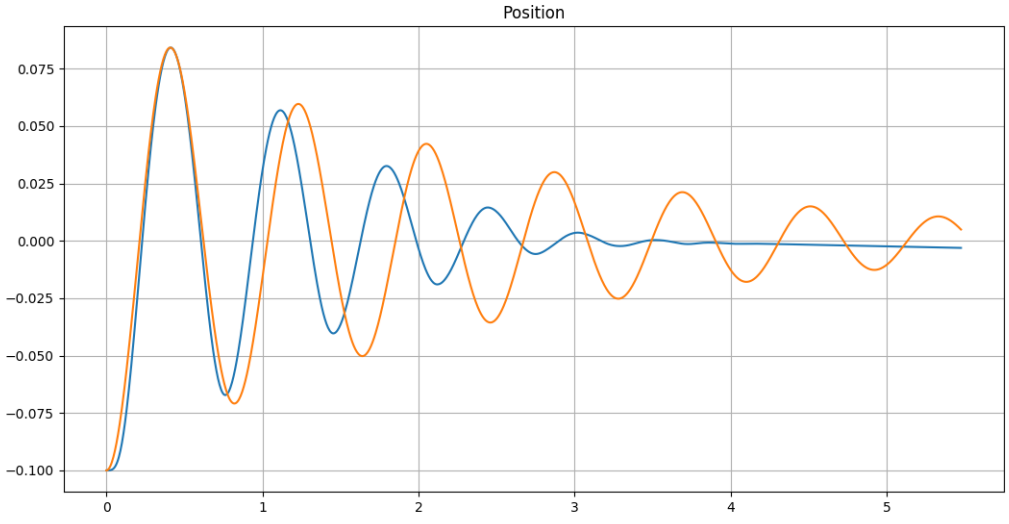
$$x(t) = -e_0 e^{-\xi\omega_0 t}\left(\cos(\omega t) + \frac{\xi\omega_0}{\omega}\sin(\omega t)\right) \text{ avec } \omega = \omega_0\sqrt{1-\xi^2}$$

Exercice XVI.14
Prouver que $x'(t) = \dfrac{e_0\omega_0}{\sqrt{1-\xi^2}}e^{-\xi\omega_0 t}\sin(\omega t)$.
En déduire les valeurs de t *et de* x *aux extremums.*

On va donc calculer les valeurs du temps, t_1, et de la position au premier extremum après l'origine, e_1, on remarque que cela correspond au maximum de la courbe.
On a alors $\omega_0 = \dfrac{\pi}{t_1\sqrt{1-\xi^2}}$ et $\dfrac{\xi}{\sqrt{1-\xi^2}} = \dfrac{-1}{\pi}\ln\left(\dfrac{e_1}{e_0}\right)$.

Exercice XVI.15
Écrire une fonction qui calcule l'indice (le premier) du maximum d'une liste.
En déduire les valeurs de t_1 *et* e_1 *puis de* ξ *et* ω_0.
`pi` *et* `log` *(pour* ln*) seront importés du module* `math`.

Voici le résultat obtenu :



Exercice XVI.16 — Question ouverte
Comment améliorer les résultats ?

6 Solutions

Solution de l'exercice XVI.1 -

Solution de l'exercice XVI.2 -

```
def traduction(ch):
    """Entrée : une chaîne de caractères représentant
                une suite de nombres séparés par des
                   tabulations
                et terminant par '\n'
       Sortie : la liste de ces nombres"""
    ch1 = ch[:-1] # On enlève le caractère '\n' final
    l1 = ch1.split('\t') # On découpe
    l2 = []
    n = len(l1)
    for i in range(n):           # Les éléments
        l2.append(float(l1[i])) # sont sont convertis en
           flottants
    return l2
```

On peut aussi faire le travail à la main.

```
def traduction(ch):
    """Entrée : une chaîne de caractères représentant
                une suite de nombres séparés par des
                   tabulations
                et terminant par '\n'
       Sortie : la liste de ces nombres"""
    l = []
    morceau = ''
    for car in ch: # On lit les caractères
        if car == '\t' or car == '\n': # fin d'un morceau
            l.append(float(morceau)) # on covertir en nombre
            morceau = '' # on re-initialise le morceau
        else:
            morceau = morceau + car # on ajoute le caractère
    return l
```

Solution de l'exercice XVI.3 -

```
liste = []
n = len(listeBrute)
for i in range(n):
    liste.append(traduction(listeBrute[i]))
```

ou `liste = [traduction(ch) for ch in listeBrute]`

Solution de l'exercice XVI.4 -

```
def extraire(k,liste):
    """Entrée : un entier et une liste
       Requis : les éléments de la liste sont des listes
                de longueur au moins k
       Sortie : la liste des k-ièmes composantes"""
    resultat  = []
    for l in liste:
        resultat.append(l[k-1])
    return resultat
```

On peut aussi utiliser les raccourcis python

```
def extraire(k,liste):
    return [l[k-1] for l in liste]
```

Solution de l'exercice XVI.5 -

- Pour la première colonne les valeurs sont affines par rapport à l'indice.
- La deuxième colonne semble être du bruit
- La troisième et la quatrième colonnes montrent des oscillations, un peu bruitées.

Solution de l'exercice XVI.6 - On choisit de renvoyer la longueur de la liste si elle ne diffère jamais de plus d'écart depuis sa position initiale.

```
def rechZero(liste,ecart):
    """Entrée : une liste et un nombre
       Sortie : le premier indice en lequel
                la liste diffère de liste[0]
                d'au moins ecart"""
    init = liste[0]
    n = len(liste)
    for i in range(n):
        if abs(liste[i] - init) >= ecart:
            return(i)
    return n
```

Solution de l'exercice XVI.7 -

```
f = 200
Delta_t = 1/f
x0 = tempsBrut[i0]
n = len(tempsBrut)
temps = []
for i in range(i0, n):
    x = tempsBrut[i]
    t = (x-x0)*Delta_t
    temps.append(t)
```

La boucle peut être remplacée par `temps = [(x-x0)*Delta_t for x in tempsBrut[i0:]]`

Solution de l'exercice XVI.8 -

```
def moy(liste):
    """Entrée : une liste non vide
       Sortie : la moyenne des termes"""
    n = len(liste)
    som = 0
    for i in range(n):
        som = som + liste[i]
    return som/n
```

```
Vg = moy(accBrute[0:i0])
```

Solution de l'exercice XVI.9 - On a $\alpha V_0 + \beta = 0$ et $\alpha V_g + \beta = g$

```
v0 = 4.8291
g = 9.81

alpha = g/(Vg - V0)
beta = -alpha*V0
```

Solution de l'exercice XVI.10 -

```
acc = []
for i in range(i0, n):
    aV = accBrute[i]
    a = alpha*aV + beta - g
    acc.append(a)
```

ou, en utilisant la définition de listes par compréhension,

```
acc = [alpha*aV + beta - g for aV in accBrute[i0:]]
```

Solution de l'exercice XVI.11 -

```
def primitive(listeFn, dt, y0):
    """Entrée : une liste de nombres et deux nombres
       Sortie : la liste des valeurs des primitives"""
    prim = [0]*n
    prim[0] = y0
    n = len(listeFn)
    for i in range(n-1): # il reste n-1 valeurs à calculer
        prim[i+1] = prim[i] + dt*listeFn[i]
    return prim
```

Solution de l'exercice XVI.12 -

```
vit = primitive(acc, Delta_t, 0)
pos = primitive(vit, Delta_t, -0.1)

plt.subplot(3,1,1)
plt.title("Accélération")
plt.grid(True)
plt.plot(temps,acc)
plt.subplot(3,1,2)
plt.title("Vitesse")
```

```
plt.grid(True)
plt.plot(temps,vit)
plt.subplot(3,1,3)
plt.title("Position")
plt.grid(True)
plt.plot(temps,pos)
plt.show()
```

Solution de l'exercice XVI.13 -

```
fichier = open("moto1.txt", "w')
n = len(pos)
for i in range(n):
    x = str(temps[i]) + "\t" + str(pos[i]) + "\n"
    fichier.write(x)
fichier.close()
```

Solution de l'exercice XVI.14 -

$x'(t) = -e_0 e^{-\xi\omega_0 t}\left(-\xi\omega_0\cos(\omega t) + \dfrac{-\xi^2\omega_0^2}{\omega}\sin(\omega t) - \omega\sin(\omega t) + \xi\omega_0\cos(t)\right)$.

Or $\dfrac{\xi^2\omega_0^2}{\omega} + \omega = \dfrac{\xi^2\omega_0^2 + \omega^2}{\omega} = \dfrac{\xi^2\omega_0^2 + \omega_0^2(1-\xi^2)}{\omega} = \dfrac{\omega_0^2}{\omega} = \dfrac{\omega_0}{\sqrt{1-\xi^2}}$.

On voit donc que les extremums sont atteints aux temps $t_k = \dfrac{k\pi}{\omega}$ et leurs valeurs sont de la forme $-(-1)^k e_0 e^{-\xi\omega_0 t_k}$.

Solution de l'exercice XVI.15 -

```
def indMax(liste):
    indMax = 0
    n = len(liste)
    for k in range(n):
        if liste[k] > liste[indMax]:
            indMax =k
    return indMax
```

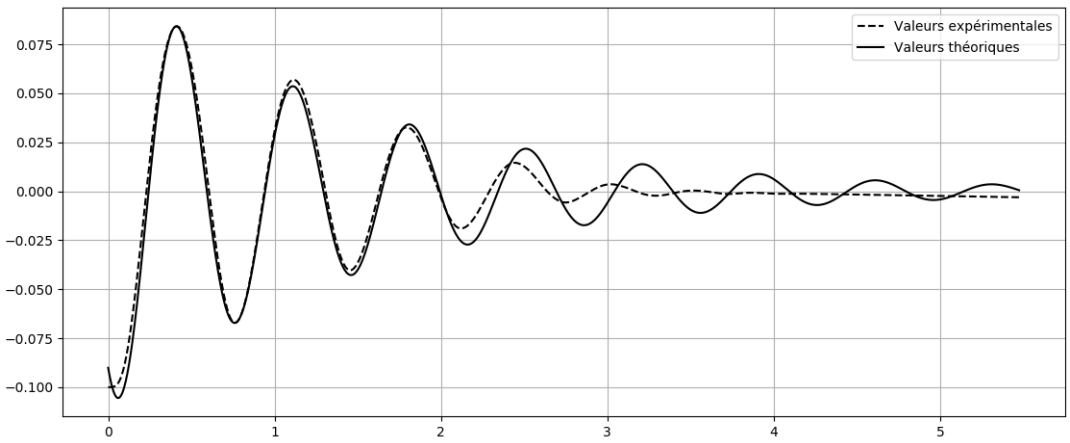
```
i1 = indMax(pos)
t1 = temps[i1]
e1 = pos[i1]
q = -log(e_1/e_0)/pi
xi = q/(pi**2+q**2)**0.5
omega0 = pi/t1/(1-xi**2)**0.5
```

Solution de l'exercice XVI.16 - On peut penser que le calcul de l'origine n'est pas idéal et espérer calculer les constantes à l'aide des deux premiers extremums. t_1 est remplacé par le temps entre les deux extremums et on calcule le quotient entre les valeurs extrémales.
On change aussi la fonction solution pour qu'elle prenne la valeur e_1 en t_1.

```
i1 = indMax(pos)
i2 = indMinDepuis(pos, i1)
t1 = temps[i1]
t2 = temps[i2]
t = (t2-t1)
omega = pi/t
e1 = abs(pos[i1])
e2 = abs(pos[i2])
q = log(e1/e2)/pi
xi = q/(1+q**2)**0.5
omega0 = pi/t/(1-xi**2)**0.5

sol = [solution(t) for t in temps]
plt.plot(temps,pos,linestyle = "dashed", color = "black",label
    ="Valeurs expérimentales")
plt.plot(temps, sol,color = "black",label="Valeurs théoriques"
    )
plt.grid()
plt.legend()
plt.show()
```

C'est moins pire



TP XVII

Méthode de Monte-Carlo

Ce T.P. a pour but

- de définir la distribution d'un grand nombre de données et la présentation associée, un histogramme,
- d'utiliser la méthode de Monte-Carlo pour illustrer la transmission des incertitudes lors de calculs.

1 Histogrammes

En statistique, un histogramme est une représentation graphique permettant de représenter la répartition d'une variable continue en la représentant avec des colonnes verticales. (Wikipedia)
Nous partons d'une liste de valeurs qui représentent les variations d'une mesure :

- les diamètres de pièces usinées,
- les tailles des enfants de 4 ans,
- les luminosités des pixels d'une photographie,
- les différentes mesures d'un pH lors d'un T.P. ...

1.1 Premières mesures

La première approche est de calculer

1. la moyenne pour obtenir une valeur centrale,
2. l'écart-type (racine de la variance) pour obtenir un ordre de grandeur de la dispersion autour de la moyenne
3. le maximum et le minimum pour obtenir un encadrement des valeurs.

Exercice XVII.1
Écrire une fonction `moyenne(liste)` *qui calcule la moyenne d'une liste.*

Exercice XVII.2
Écrire une fonction `ecart_type(liste)` *qui calcule l'écart-type d'une liste.*
Rappel : la variance d'une liste est la moyenne des $(x_i - m)^2$ *où* m *est la moyenne des* x_i *c'est aussi la moyenne des carrés,* x_i^2*, à laquelle on soustrait* m^2*.*

Exercice XVII.3
Écrire une fonction `maximum(liste)`*, resp.* `minimum(liste)`*, qui calcule le maximum, resp.le minimum, des termes d'une liste. On pourra supposer que la liste est non vide.*

1.2 Comptage

On peut aller plus loin en découpant un intervalle contenant les valeurs en plusieurs sous-intervalles (on parle de **bins**) et comptant le nombre de valeurs de la liste dans chacun de ces intervalles.
On choisit la méthode suivante :

1. on choisit le nombre d'intervalles, n
2. on calcule le minimum, `mini`, et le maximum, `maxi`, des valeurs de la liste
3. on détermine les intervalles, $I_0 = [a_0; a_1[,\ I_1 = [a_1; a_2[,\ \cdots I_{n-1} = [a_{n-1}; a_n]$ avec $a_k = \texttt{mini} + k.h$ et $h = \frac{1}{n}\big(\texttt{maxi} - \texttt{mini}\big)$,
4. on crée une liste de taille n initialisée à 0, `histo`,
5. pour chaque valeur x, on détermine l'intervalle auquel x appartient, I_k, et on incrémente la valeur de `histo[k]` de 1.

On remarque que x appartient à I_k si et seulement si $a_k \leqslant x < a_{k+1}$, c'est-à-dire $k.h \leqslant x - \texttt{mini} < (k+1).h$ d'où $k = \left\lfloor \dfrac{x - \texttt{mini}}{h} \right\rfloor$. Il y a cependant un cas particulier : si $x = \texttt{maxi}$ alors $\left\lfloor \dfrac{x - \texttt{mini}}{h} \right\rfloor$ vaut n mais il faut compter x dans I_{n-1}.

Exercice XVII.4
Écrire une fonction `comptage(liste, n)` *qui renvoie le nombre d'éléments par intervalle dans une liste* `histo` *selon l'algorithme ci-dessus ainsi que le pas h et la valeur du minimum.*

Dans le cas `L0 = [4.3, 5.1, 7.8, 5.4, 4.4, 7.3, 6.8, 6.7, 5.9, 7.1]`,
`comptage(L0, 5)` renvoie `([2, 2, 1, 2, 3], 0.7, 4.3)`.
Un fonction semblable existe dans le module `numpy` : `np.histogram`

1.3 Affichage

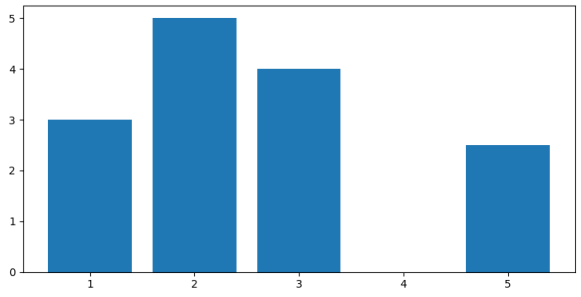
Pour obtenir la représentation graphique des valeurs calculées, on utilise un diagramme en barres disponible avec la fonction `plt.bar` où `plt` est le nom donné au module graphique de Python.

```
import matplotlib.pyplot as plt
```

Cette fonction a deux paramètres obligatoires et de nombreux paramètres optionnels (nommés) dont nous n'allons utiliser qu'un seul : `width`.
`plt.bar(X, Y, width = w)` reçoit 3 paramètres

1. `X` est la liste des centres des intervalles I_k définis ci-dessus,
2. `Y` est la liste, de même longueur que `X`, des nombres d'éléments dans chaque intervalle,
3. `w` est la largeur.

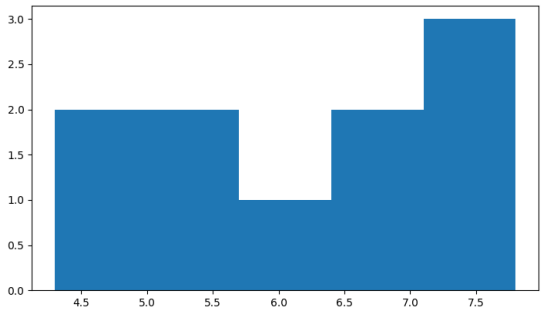
Si n est la longueur des listes `X` et `Y`, la fonction va tracer n rectangles pour $0 \leqslant i < n$, dont la base est le segment $\big[X[i] - w/2; X[i] + w/2\big]$ sur l'axe des abscisses et dont la hauteur est $Y[i]$.
Les points $X[i]$ ne sont pas obligatoirement régulièrement espacés et les $Y[i]$ peuvent être des réels.
Par exemple `plt.bar([1, 2, 3, 5], [3, 5, 4, 2.5], width = 0.8)` donne



On peut donc écrire la fonction

```
def histogramme(liste, n):
    valeurs, h, mini = comptage(liste, n)
    centres = ....
    for i in range(n):
        ....
    plt.bar(centres, valeurs, width = h)
    plt.show()
```

`histogramme(L0, 5)` donne le résultat suivant.



Exercice XVII.5
Compléter le code ci-dessus pour calculer la liste des centres.

Cette fonction est semblable à la fonction `plt.hist(liste, bins = n)`

2 Simulation des incertitudes

Nous allons utiliser la représentation par histogrammes pour illustrer la propagation des incertitudes simples.

2.1 Incertitude d'une valeur

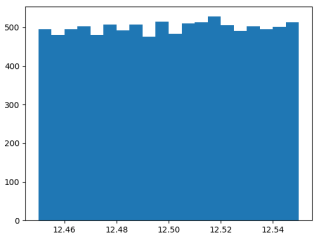
On considère, par exemple, un volume de 12,5 ml mesuré à la pipette graduée ; on sait le volume réel est compris entre 12,45 ml et 12,55 ml, l'incertitude dépend entre autres de la dextérité du manipulateur.
On va simuler cette incertitude en produisant une liste de valeurs aléatoirement choisies dans l'intervalle. Pour cela on utilisera la fonction `rd.random` du module `random`

```
import random as rd
```

`rd.random()` renvoie un flottant choisi aléatoirement dans l'intervalle $[0; 1[$.

Exercice XVII.6
Écrire une fonction `liste_alea(x0, e, n)` *qui renvoie une liste de taille n dont les valeurs sont choisies aléatoirement dans l'intervalle $[x_0 - e; x_0 + e]$.*

Si on représente l'histogramme de la liste `liste_alea(12.5, 0.05, 10000)` avec 20 intervalles, on obtient des valeurs presque égales.



2.2 Opérations

Exercice XVII.7
On se donne deux simulations `A = liste_alea(12.5, 0.05, 10000)` *et* `B = liste_alea(8.3, 0.05, 10000)`. *Déterminer la liste* `X` *telle que* `X[i] = A[i] + B[i]` *et représenter son histogramme avec 20 intervalles.*

Exercice XVII.8
On se donne deux simulations `A = liste_alea(12.5, 0.05, 10000)` *et* `B = liste_alea(8.32, 0.005, 10000)`. *Déterminer la liste* `X` *telle que* `X[i] = A[i] + B[i]` *et représenter son histogramme avec 20 intervalles.*
Comparer avec le cas précédent, commentez.

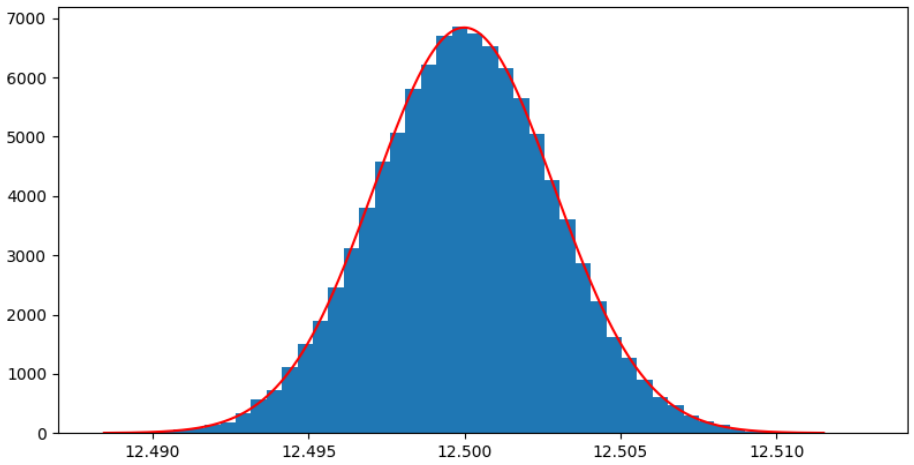
Exercice XVII.9
Calculer des simulations sur 10000 points de valeurs $a = 12,5$, $b = 8,3$, $c = 4,7$ *et* $d = 13,4$, *chacune avec une précision* $\pm 0,05$.
Calculer et représenter les valeurs de $x = \dfrac{a+b}{c+d}$.

2.3 Vers une gaussienne

Un comportement souvent observé est que si on combine plusieurs incertitudes on s'approche d'une gaussienne.

Exercice XVII.10
1. *Créer une liste de taille 100 dont les éléments sont des simulations sur 100000 points d'une valeur* $a = 12,5$ *avec une précision* $\pm 0,05$. *On obtient une liste de listes de taille* $100 \times 100\,000$.
2. *Calculer et représenter les valeurs de la liste* `M` *de taille* $100\,000$ *telle que* `M[i]` *est la moyenne des* `X[k][i]` *pour* k *variant de 0 à 99.*
3. *Calculer la moyenne* m *et l'écart-type* σ *de* `M`.

On pourra comparer avec une gaussienne sur le même graphe.
La fonction est de la forme $x \mapsto \dfrac{K}{\sqrt{2\pi}\sigma} \exp\left(\dfrac{-(x-m)^2}{2\sigma^2}\right)$ où K est la surface de l'histogramme : c'est le nombre de points multiplié par h, la largeur des intervalles.



3 Solutions

Solution de l'exercice XVII.1 -

```
def moyenne(liste):
    n = len(liste)
    s = 0
    for x in liste:
        s = s + x
    return s/n
```

Solution de l'exercice XVII.2 -

```
def ecart_type(liste):
    n = len(liste)
    m = moyenne(liste)
    s = 0
    for x in liste:
        s = s + (x-m)**2
    return (s/n)**0.5
```

Solution de l'exercice XVII.3 -

```
def maximum(liste):
    maxi = liste[0]
    for x in liste:
        if x > maxi:
            maxi = x
    return maxi
```

```
def minimum(liste):
    mini = liste[0]
    for x in liste:
        if x < mini:
            mini = x
    return mini
```

Solution de l'exercice XVII.4 -

```
def comptage(liste, n):
    mini = minimum(liste)
    maxi = maximum(liste)
    h = (maxi - mini)/n
    histo = [0]*n
    for x  in liste:
        k = int((x-mini)/h)
        if k == n:
            k = n - 1
        histo[k] += 1
    return histo, h, mini
```

Solution de l'exercice XVII.5 -

```
def histogramme(liste, n):
    valeurs, h, mini = comptage(liste, n)
    centres = [0]*n
    for i in range(n):
        centres[i] = mini + (i+1/2)*h
    plt.bar(centres, valeurs, width = h)
    plt.show()
```

Solution de l'exercice XVII.6 -

```
def liste_alea(x0, e, n):
    out = [0]*n
    for i in range(n):
        x = x0 + (2*rd.random() - 1)*e
        out[i] = x
    return out
```

Solution de l'exercice XVII.7 -

```
A = liste_alea(12.5, 0.05, 10000)
B = liste_alea(8.3, 0.05, 10000)
X = [A[i] + B[i] for i in range(len(A))]
histogramme(X, 20)
```

Solution de l'exercice XVII.8 -

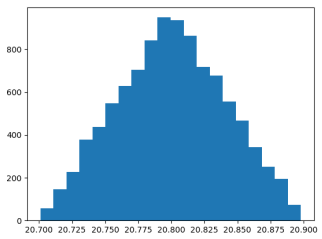
```
A = liste_alea(12.5, 0.05, 10000)
B = liste_alea(8.32, 0.005, 10000)
X = [A[i] + B[i] for i in range(len(A))]
histogramme(X, 20)
```

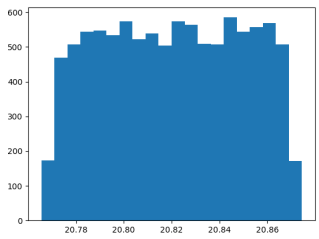
Les 2 incertitudes ne se combinent pas pour former une distribution triangulaire car l'une est petite devant l'autre. On peut considérer qu'il n'y a que l'incertitude sur A.

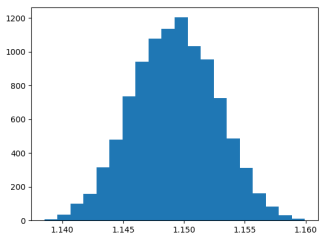
Solution de l'exercice XVII.9 -

```
A = liste_alea(12.5, 0.05, 10000)
B = liste_alea(8.3, 0.05, 10000)
C = liste_alea(4.7, 0.05, 10000)
D = liste_alea(13.4, 0.05, 10000)
X = [(A[i] + B[i])/(C[i]+D[i]) for i in range(len(A))]
histogramme(X, 20)
```

On obtient une distribution qui ressemble à une gaussienne.







Solution de l'exercice XVII.10 -

```
import math
N = 100000
N1 = 100
A = [0]*N1
for i in range(N1):
    A[i] = liste_alea(12.5, 0.05, N)

M = [0]*N
for i in range(N):
    s = 0
    for k in range(N1):
        s = s + A[k][i]
    M[k] = s/N1

def gauss(x, m, s):
    return math.exp(-(x-m)**2/2/s**2)/s/(2*math.pi)**0.5

def histogramme_gauss(liste, n):
    valeurs, h, mini = comptage(liste, n)
    centres = [0]*n
    for i in range(n):
        centres[i] = mini + (i+1/2)*h
    plt.bar(centres, valeurs, width = h)
    m = moyenne(M)
    s = ecart_type(M)
    X = [m - 4*s + i*8*s/1000 for i in range(1000)]
    Y = [N*h*gauss(x, m, s) for x in X]
    plt.plot(X, Y, color="red")
    plt.show()


histogramme_gauss(M, 50)
```

TP XVIII

Intégration I : point milieu

Ce T.P. a pour but d'étudier l'erreur commise lors du calcul d'une valeur approchée d'une intégrale. On y comparera les sommes de Riemann (à gauche) avec une méthode qui n'a peut être pas été vue en cours. La démarche se généralise à toutes les méthodes, en particulier la méthode du point milieu a une efficacité semblable à la méthode du trapèze. Nous utiliserons les fonctions mathématiques du module `math`.

```
from math import *
```

1 Définition

Les méthodes de calcul d'une intégrale $\int_a^b f(t)\mathrm{d}t$ consistent à découper l'intervalle $[a;b]$ en n intervalles $[x_k;x_{k+1}]$, $0 \leqslant k < n$ avec $x_k = a + k\frac{b-a}{n}$ pour $0 \leqslant k \leqslant n$ puis à approcher l'intégrale $\int_{x_k}^{x_{k+1}} f(t)\mathrm{d}t$ par une expression simple :

1. $(x_{k+1} - x_k)f(x_k)$ pour la méthode des rectangles à gauche,
2. $(x_{k+1} - x_k)f(x_{k+1})$ pour la méthode des rectangles à droite,
3. $(x_{k+1} - x_k)\dfrac{f(x_k) + f(x_{k+1})}{2}$ pour la méthode des trapèzes

Voici une écriture du calcul avec la méthode des rectangles à gauche :

```
def rect_g(f, a, b, n):
    pas = (b-a)/n
    s = 0
    for k in range(n):
        c = a + k*pas
        s = s + f(c)
    return s*pas
```

Nous allons implémenter ici une autre méthode, nommée "du point milieu" :

$$\int_{x_k}^{x_{k+1}} f(t)\mathrm{d}t \simeq (x_{k+1} - x_k) f\left(\frac{x_k + x_{k+1}}{2}\right)$$

Exercice XVIII.1 — Écriture
Écrire la fonction `point_m(f, a, b, n)` *qui approche l'intégrale de f entre a et b par cette méthode en subdivisant $[a; b]$ en n intervalles.*

2 Calculs approchés de π

Nous allons approcher la valeur de $I = \displaystyle\int_0^1 \frac{4}{1+t^2} dt = \pi$.

Exercice XVIII.2
Créer la fonction $f_1 : t \mapsto \dfrac{4}{1+t^2}$

L'erreur commise pour n est l'écart entre π est la valeur calculée :
`abs(rect_g(f1, 0, 1, n) - pi)` ou `abs(point_m(f1, 0, 1, n) - pi)`

Exercice XVIII.3
Calculer les erreurs pour $n = 10^p$ avec $p \in \{1, 2, 3, 4, 5, 6, 7\}$ (le dernier calcul peut être long) avec chacune des deux méthodes.
Que peut-on dire de cette erreur lorsque n est multiplié par 10 ?

3 Erreur comme fonction de n

On souhaite maintenant étudier plus précisément l'erreur en fonction de n.

Exercice XVIII.4
Définir une liste d'entiers `N` *prenant 100 valeurs de 5000 à 500000 espacées de 5000 :*
`N = [5000, 10000, 15000, ..., 495000, 500000]`
Définir la liste `err_g` *correspondant aux erreurs de la méthode des rectangles pour le calcul de I.*
Définir la liste `err_m` *correspondant aux erreurs de la méthode du point milieu pour le calcul de I.*
Le calcul peut être un peu long (10 secondes).

On peut alors représenter le graphe de l'erreur en fonction de n.
On utilise le module `matplotlib`, à importer.

```
import matplotlib.pyplot as plt
```

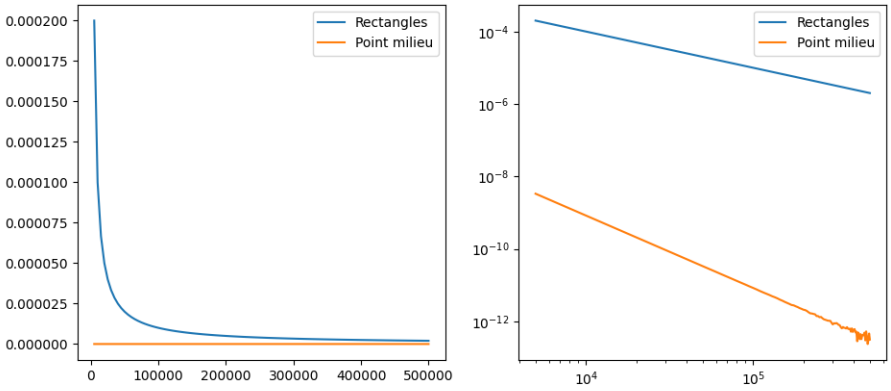
`plt.plot(X, Y)` calcule le graphe, ce sont les segments de droite joignant les points de coordonnées `(X[i], Y[i])` et `(X[i+1], Y[i+1])`. La fonction demande deux listes de même taille [1].
`plt.show()` dessine alors le graphe à l'écran.

```
plt.plot(N, err_g, label = "Rectangles")
plt.plot(N, err_m, label = "Point milieu")
plt.legend()
plt.show()
```

Ici on peut conjecturer que l'erreur est de la forme $e = \frac{K}{n^p}$. Pour le mettre en évidence on remarque que $\ln(e) = \ln(K) - p\ln(n)$; une échelle logarithmique devrait donner une droite de pente $-p$.
On impose une échelle logarithmique avec les instruction `plt.xscale('log')` et `plt.yscale('log')`

1. `plt.plot(Y)` est possible, l'abscisse est alors déterminée par l'indice.

```
plt.xscale('log')
plt.yscale('log')
plt.plot(N, err_g, label = "Rectangles")
plt.plot(N, err_m, label = "Point milieu")
plt.legend()
plt.show()
```



Exercice XVIII.5

À quoi sont dues les variations pour les grandes valeurs de n dans le cas du point milieu ?

On peut conjecturer que la pente est -1, l'erreur serait de la forme $\dfrac{K_1}{n}$ pour la méthode des rectangles et une pente -2 avec une erreur de la forme $\dfrac{K_2}{n^2}$ pour la méthode du point milieu.
Pour le vérifier, on peut calculer la liste des erreurs multipliées par n^p.

Exercice XVIII.6

Calculer et représenter les listes `K_g` *et* `K_m` *telles que*
`K_g[i] = err_g[i]*N[i]` *et* `K_m[i] = err_m[i]*N[i]**2`.

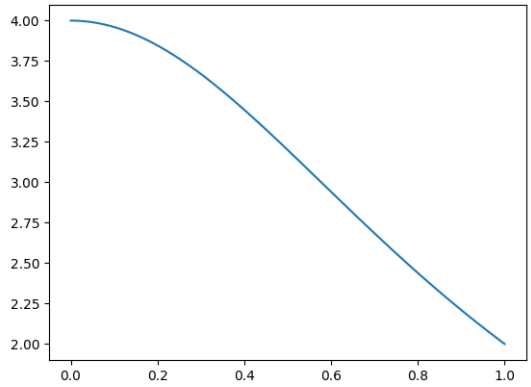
4 Un résultat contre-intuitif

Dans cette partie on ne suppose plus que la fonction à intégrer est donnée sous forme d'une évaluation en tout point par une fonction Python mais qu'elle n'est connue que par ses valeurs en N points, on dit qu'elle est tabulée. Ce sera souvent le cas pour des fonctions calculées expérimentalement : un processus d'acquisition a enregistré les valeurs selon une période donnée.

Exercice XVIII.7

Écrire les instructions qui permettent de calculer la liste `F1` *des valeurs des $f_1(t_k)$ pour $t_k = k.10^{-3}$, $0 \leqslant k \leqslant 1000$;* `F1` *sera de longueur 1001.*
On calculera aussi la liste `T` *des t_k.*

Ces listes permettent de tracer le graphe de f_1 : `plt.plot(T, F1)`.



Le calcul d'une valeur approchée de l'intégrale est maintenant unique pour chaque méthode.
Si $T = [t_0, t_1, \ldots, t_{n-1}]$ est la liste des abscisses et si $Y = [y_0, y_1, \ldots, y_{n-1}]$ est la liste des ordonnées avec $F(t_k) = y_k$ alors, pour la méthode des rectangles à gauche,

$$\int_{t_0}^{t_{n-1}} f(t)\mathrm{d}t \simeq \sum_{k=0}^{n-2} f(t_k)(t_{k+1} - t_k) = (t_1 - t_0)\sum_{k=0}^{n-2} y_k$$

Ici on a supposé que le pas $t_{k+1} - t_k$ était constant, ce qui sera vrai.

Exercice XVIII.8
Écrire une fonction `integrale_rg(T, Y)` *qui calcule l'intégrale de la fonction représentée par* `Y` *sur* $[a; b]$ *où* a *et* b *sont les valeurs extrêmes de la liste* `T` *supposée à pas constant.*
Quel est l'erreur commise pour le calcul de I *?*

Dans le cas de la méthode du point milieu, comme on a besoin de la valeur de f au milieu de l'intervalle, on va découper l'intégrale en somme de termes de la forme $\int_{t_{2k}}^{t_{2(k+1)}} f(t)\mathrm{d}t$ que l'on approche par $f(t_{2k+1})(t_{2(k+1)} - t_{2k}) = f(t_{2k+1})(t_2 - t_0)$.
On ne peut utiliser cette méthode que si la **longueur est impaire**.

Exercice XVIII.9
Écrire une fonction `integrale_pm(T, Y)` *qui calcule l'intégrale de la fonction représentée par* `Y` *sur* $[a; b]$ *où* a *et* b *sont les valeurs extrêmes de la liste* `T` *supposée à pas constant et de longueur impaire. On emploiera la méthode du point milieu.*
Quel est l'erreur commise pour le calcul de I *?*

On aboutit à un résultat paradoxal : on obtient une bien meilleure approximation en employant seulement la moitié des points !

4.1 Complément

En général on emploie plutôt la méthode des trapèzes qui donne un résultat un peu meilleur que le point milieu.

Exercice XVIII.10
Écrire une fonction `integrale_tr(T, Y)` *qui calcule l'intégrale de la fonction représentée par* `Y` *sur* $[a; b]$ *où* a *et* b *sont les valeurs extrêmes de la liste* `T` *supposée à pas constant avec la méthode des trapèzes.*
Quel est l'erreur commise pour le calcul de I *?*

5 Solutions

Solution de l'exercice XVIII.1 -

```
def point_m(f, a, b, n):
    pas = (b-a)/n
    s = 0
    for k in range(n):
        c = a + (k + 1/2)*pas
        s = s + f(c)
    return s*pas
```

Solution de l'exercice XVIII.2 -

```
def f1(t):
    return 4/(1+t**2)
```

Solution de l'exercice XVIII.3 -

```
for p in range(1, 8):
    n = 10**p
    e1 = abs(rect_g(f1, 0, 1, n) - pi)
    e2 = abs(point_m(f1, 0, 1, n) - pi)
    print("Pour n = {}, l'erreur est : ".format(n))
    print("  {:7.1e} pour rect_g, {:7.1e} pour point_m".format
       (e1, e2))
```

```
Pour n = 10, l'erreur est :
  9.8e-02 pour rect_g, 8.3e-04 pour point_m
Pour n = 100, l'erreur est :
  1.0e-02 pour rect_g, 8.3e-06 pour point_m
Pour n = 1000, l'erreur est :
  1.0e-03 pour rect_g, 8.3e-08 pour point_m
Pour n = 10000, l'erreur est :
  1.0e-04 pour rect_g, 8.3e-10 pour point_m
Pour n = 100000, l'erreur est :
  1.0e-05 pour rect_g, 8.4e-12 pour point_m
Pour n = 1000000, l'erreur est :
  1.0e-06 pour rect_g, 2.9e-14 pour point_m
Pour n = 10000000, l'erreur est :
  1.0e-07 pour rect_g, 6.2e-14 pour point_m
```

Pour les premières valeurs de n, l'erreur est divisée par 10 (resp. par 100) chaque fois que n est multiplié par 10 pour la méthode des rectangles (resp. du point milieu).
Dans le cas du point milieu, il n'y a plus d'amélioration à la fin.

Solution de l'exercice XVIII.4 -

```
taille = 100
pas = 5000
N  = [0]*taille
err_g = [0]*taille
err_m = [0]*taille
for i in range(taille):
    N[i] = (i+1)*pas
    err_g[i] = abs(pi -  rect_g(f1, 0, 1, N[i]))
```

```
    err_m[i] = abs(pi -  point_m(f1, 0, 1, N[i]))
```

On peut utiliser les sucres syntactiques de Python :

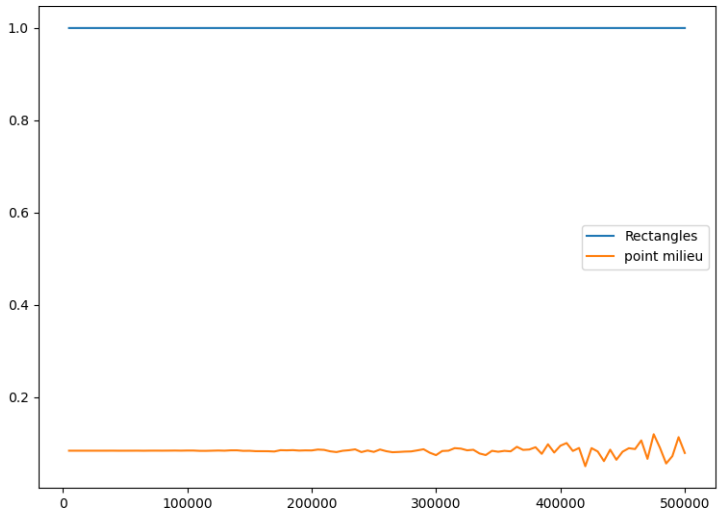
```
taille = 100
pas = 5000
N  = [(i+1)*pas for i in range(taille)]
err_g = [abs(pi - rect_g(f1, 0, 1, n)) for n in N]
err_m = [abs(pi - point_m(f1, 0, 1, n)) for n in N]
```

Solution de l'exercice XVIII.5 -
On fait une somme de 10^5 termes, chacun étant approché avec une précision de 10^{-17} ; on aboutit à une erreur de l'ordre de 10^{-12} qui est l'approximation mathématique. On ne pourra plus améliorer la précision.

Solution de l'exercice XVIII.6 -

```
n = len(N)
K_g = [0]*n
K_m = [0]*n
for i in range(n):
    K_g[i] = err_g[i]*N[i]
    K_m[i] = err_m[i]*N[i]**2

plt.plot(N, K_g, label = "Rectangles")
plt.plot(N, K_m, label = "point milieu")
plt.legend()
plt.show()
```



Solution de l'exercice XVIII.7 -

```
N1 = 1001
T = [0]*N1
F1 = [0]*N1
for k in range(N1):
```

```
    T[k] = k/(N1 - 1)
    F1[k] = f1(T[k])
```

Solution de l'exercice XVIII.8 -

```
def integrale_rg(T, Y):
    s = 0
    n = len(T)
    pas = T[1] - T[0]
    for k in range(n-1):
        s = s + Y[k]
    return s*pas

print(abs(pi - integrale_rg(T, F1)))
```

L'erreur de de l'ordre de 10^{-3}.

Solution de l'exercice XVIII.9 -

```
def integrale_pm(T, Y):
    s = 0
    n = len(T)
    pas = T[2] - T[0]
    for k in range(1, n, 2):
        s = s + Y[k]
    return s*pas

print(abs(pi - integrale_pm(T, F1)))
```

L'erreur de de l'ordre de 4.10^{-7}.

Solution de l'exercice XVIII.10 -

```
def integrale_tr(T, Y):
    s = (Y[-1] - Y[0])/2
    n = len(T)
    pas = T[1] - T[0]
    for k in range(n-1):
        s = s + Y[k]
    return s*pas

print(abs(pi - integrale_tr(T, F1)))
```

L'erreur de de l'ordre de 2.10^{-7}.

TP XIX

Intégration II : Fourier

Un problème important lorsque l'on étudie des fonctions est de définir, de manière approchée, une fonction avec un nombre limité de variables. En effet une fonction n'est connue que par sa valeur en tout point d'un intervalle I (souvent $\mathbb{R}$ ou un segment), c'est un nombre infini (même non dénombrable) de variables mais un ordinateur ne peut n'en gérer qu'un nombre fini.
Il existe deux stratégies principales.

1. On se restreint à un segment $[a;b]$ qu'on subdivise en plus petit segments $[a_k;a_{k+1}]$; sur chacun d'eux on approche la fonction par une fonction **très** simple. Par exemple les valeurs (approchées) des $f(a_k)$ permettent d'approcher f par une fonction affine sur $[a_k;a_{k+1}]$. Si on ajoute la valeur des dérivées on obtient une approximation par les fonctions splines cubiques.
2. On choisit un type de fonctions considérées comme basiques sur I et on cherche à approcher f par une somme finie de telles fonctions. On utilise des approximations par des polynômes, des polynômes trigonométriques, des ondelettes ...

Dans l'étude des phénomènes périodiques on utilise la décomposition de Fourier : une fonction T-périodique est approchée par un polynôme trigonométrique de période T c'est à dire une sommes de fonctions de la forme $t \mapsto \cos(n\omega t)$ et $t \mapsto \sin(n\omega t)$ avec $\omega = \dfrac{2\pi}{T}$.

1 Définition

On admet que pour une fonction f T-périodique et assez régulière [1], les coefficients

1. $a_0 = \dfrac{1}{T}\displaystyle\int_{-T/2}^{T/2} f(t)\mathrm{d}t$, la valeur moyenne de f
2. $a_n = \dfrac{2}{T}\displaystyle\int_{-T/2}^{T/2} \cos(n\omega t)f(t)\mathrm{d}t$ pour $n \geqslant 1$,
3. $b_n = \dfrac{2}{T}\displaystyle\int_{-T/2}^{T/2} \sin(n\omega t)f(t)\mathrm{d}t$ pour $n \geqslant 1$

avec $\omega = \dfrac{2\pi}{T}$ permettent approcher f par la somme de Fourier d'ordre n

$$S_N(f)(t) = a_0 + \sum_{n=1}^{N} a_n \cos(n\omega t) + b_n \sin(n\omega t)$$

1. On admet que ce sera toujours le cas des fonctions envisagées

2 Calculs

On utilisera les fonctions mathématiques du module `math`.

```
from math import *
```

2.1 Intégrales

Exercice XIX.1
Écrire une fonction `integrale(f, a, b)` *qui calcule qui calcule une valeur approchée de l'intégrale* $\int_a^b f(t)dt$ *en utilisant la méthode des trapèzes en découpant l'intervalle en 1000 sous-intervalles de même largeur.*

On pourra aussi utiliser la fonction de calcul des intégrales du module `scipy`

```
from scipy.integrate import quad
```

1. `quad(f, a, b)` renvoie un couple (I, e) où I est une valeur approchée de $\int_a^b f(t)\mathrm{d}t$ et e est une estimation de l'erreur commise.
2. La fonction `quad` permet des bornes infinies, `inf` de la bibliothèque `math` représente $+\infty$.

On peut remplacer la fonction de l'exercice ci-dessus par

```
def integrale(f, a, b):
    I, e = quad(f, a, b)
    return I
```

2.2 Coefficients

Dans le calcul des coefficient on doit intégrer des fonctions différentes pour chaque indice ; on définira une fonction dans le corps d'une fonction.

```
def liste_a(f, N, T):
    ...
    for n in range(1, N+1)
        def fn(x):
            return f(x)*cos(omega*n*x)
        .... # On peut utiliser fn
    return
```

Exercice XIX.2
Écrire une fonction `liste_a(f, N, T)` *qui calcule la liste des coefficients* a_n, $0 \leqslant n \leqslant N$, *de la fonction* f *périodique de période* T. *On notera qu'on renvoie une liste de taille* $N+1$.

Exercice XIX.3
Faire de même pour les coefficients b_n, *on laissera la valeur 0 pour* b_0.

2.3 Sommes de Fourier

Pour éviter les calculs répétés, l'évaluation des sommes partielles reçoit en paramètres les listes, de même longueur, des coefficients a_n et b_n.

Exercice XIX.4

Écrire une fonction `somme_Fourier(A, B, T, x)` *qui calcule la somme partielle* $a_0 + \sum_{n=1}^{N} a_n \cos(n\omega x) + b_n \sin(n\omega x)$ *où* $\omega = \frac{2\pi}{T}$ *et* a_n *(resp.* b_n*) est la valeur de* `A[n]` *(resp.* `B[n]`*).*

3 Visualisations

Pour afficher le graphe d'une fonction, une méthode possible est la suivante.

1. On doit avoir importé le module graphique

```
import matplotlib.pyplot as plt
```

2. On définit une liste des abscisses pour représenter un intervalle $[a;b]$. Cela peut se faire avec

```
n = 1000
pas  = (b-a)/n
X = [a + k*pas for k in range(n+1)]
```

3. On calcule la liste des images de ces abscisses

```
X = [f(x) for x in X]
```

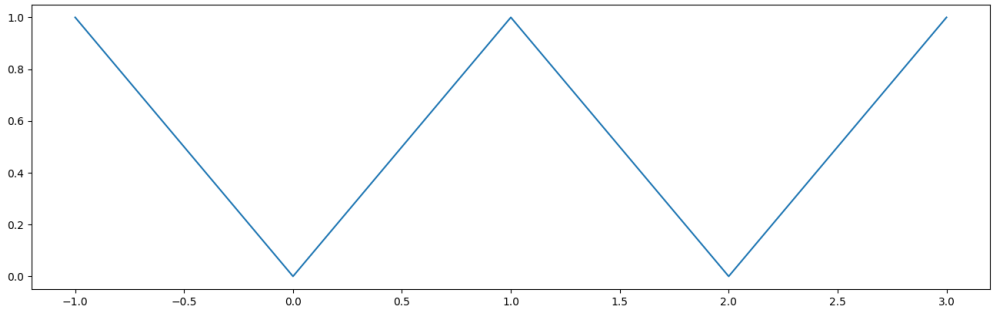
4. On trace le graphe

```
plt.plot(X, Y)
plt.show()
```

5. Si on a besoin de tracer les graphes de plusieurs fonctions, on fait un calcul du tracé (`plt.plot`) par fonction et o finit par un unique `plt.show()`. On peut légender les tracés.

```
plt.plot(X, Y1, label = "Fonction f1")
plt.plot(X, Y2, label= "Fonction f2")
plt.legend()
plt.show()
```

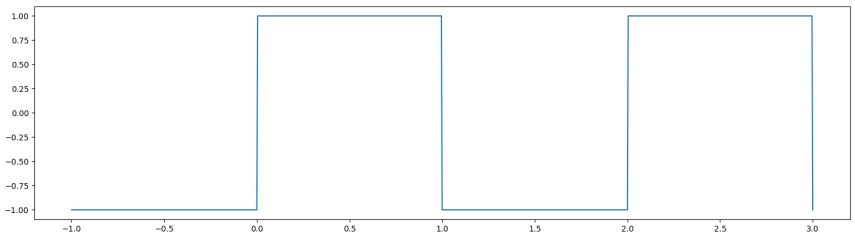
3.1 Signal triangulaire



Exercice XIX.5
Écrire la fonction `triangle(x)` *qui renvoie la valeur de la fonction 2-périodique égale à* $|x|$ *pour* $x \in [-1;1]$.

Exercice XIX.6
Représenter sur un même graphe la fonction triangle et les sommes partielles, $S_n(f)$*, de sa série de Fourier pour* $n \in \{1,2,3,5,20\}$ *sur l'intervalle* $[-1;3]$.

3.2 Signal rectangulaire



Exercice XIX.7
Écrire la fonction `rectangle(x)` *qui renvoie la valeur de la fonction 2-périodique égale à* 1 *sur* $]0;1[$*,* *à* -1 *sur* $]-1;1[$ *et à 0 aux points de* $\mathbb{Z}$*.*

Exercice XIX.8
Représenter sur un même graphe la fonction rectangle et les sommes partielles, $S_n(f)$*, de sa série de Fourier pour* $n \in \{1, 5, 10, 20, 50\}$ *sur l'intervalle* $[-1;3]$*.*

On voit apparaître un dépassement incompressible de la valeur des sommes par rapport aux créneaux, c'est le phénomène de Gibbs.
On peut le faire disparaître en utilisant les sommes de Fejer, ce sont les moyennes des sommes de Fourier :

$$F_N(f)(t) = a_0 + \sum_{n=1}^{N} \big(a_n \cos(n\omega t) + b_n \sin(n\omega t)\big) \left(1 - \frac{n}{N+1}\right)$$

Exercice XIX.9
Représenter sur un même graphe la fonction rectangle et les sommes partielles, $S_n(f)$*, de sa somme de Fejer pour* $n \in \{1, 5, 10, 20, 50\}$ *sur l'intervalle* $[-1;3]$*.*

4 Solutions

Solution de l'exercice XIX.1 -

```
def integrale(f, a, b):
    n = 1000
    pas = (b-a)/n
    somme = (f(a) + f(b))/2
    for k in range(1, n-1):
        ak = a + k*pas
        somme = somme + f(ak)
    return somme*pas
```

Solution de l'exercice XIX.2 -

```
def liste_a(f, N, T):
    A = [0]*(N+1)
    A[0] =  integrale(f, -T/2, T/2)/T
    omega = 2*pi/T
    for n in range(1, N+1):
        def fn(x):
            return f(x)*cos(omega*n*x)
        A[n] = 2*integrale(fn, -T/2, T/2)/T
    return A
```

Solution de l'exercice XIX.3 -

```
def liste_b(f, N, T):
    B = [0]*(N+1)
    omega = 2*pi/T
    for n in range(1, N+1):
        def fn(x):
            return f(x)*sin(omega*n*x)
        B[n] = 2*integrale(fn, -T/2, T/2)/T
    return B
```

Solution de l'exercice XIX.4 -

```
def somme_Fourier(A, B, T, x):
    N = len(A)
    s = 0
    w = 2*pi/T
    for n in range(N):
        s = s + A[n]*cos(n*w*x) + B[n]*sin(n*w*x)
    return s
```

Solution de l'exercice XIX.5 -
Pour $x \in [2k-1; 2k+1]$ on a $x - 2k \in [-1; 1]$ donc triangle(x) = triangle$(x-2k) = |x-2k|$.
$2k - 1 \leqslant x < 2k + 1$ est équivalent à $k \leqslant \frac{x+1}{2} \leqslant k + 1$; on peut choisir $k = \left\lfloor \frac{x+1}{2} \right\rfloor$.

```
def triangle(x):
    k = floor((x+1)/2)
    return abs(x-2*k)
```

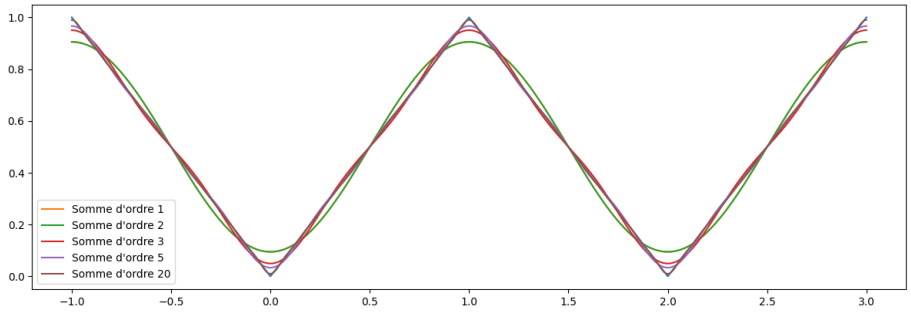
Solution de l'exercice XIX.6 -

```
a = -1
b = 3
n = 1000
pas  = (b-a)/n
X = [a + k*pas for k in range(n+1)]
Y = [triangle(x) for x in X]
plt.plot(X, Y)

A = liste_a(triangle, 20, 2)
B = liste_b(triangle, 20, 2)

for N in [1, 2, 3, 5, 20]:
    Y = [somme_Fourier(A[ : N+1], B[ : N+1], 2, x) for x in X]
    plt.plot(X, Y, label="Somme d'ordre {}".format(N))
plt.legend()
plt.show()
```

Comme la fonction est paire les coefficients b_n sont tous nuls.
De plus les coefficients a_{2p} sont nuls aussi pour $p \geqslant 1$; il n'y a pas de différence entre $S_1(f)$ et $S_2(f)$.



Solution de l'exercice XIX.7 -
Pour $x \in [2k-1; 2k+1]$ on a $x - 2k \in [-1; 1]$ donc $\text{triangle}(x) = \text{triangle}(x-2k) = |x-2k|$.
$2k - 1 \leqslant x < 2k + 1$ est équivalent à $k \leqslant \frac{x+1}{2} \leqslant k + 1$; on peut choisir $k = \left\lfloor \frac{x+1}{2} \right\rfloor$.

```
def rectangle(x):
    k = floor((x+1)/2)
    if 0 < x - 2*k < 1:
        return 1
    elif -1 < x - 2*k < 0:
        return -1
    else:
        return 0
```

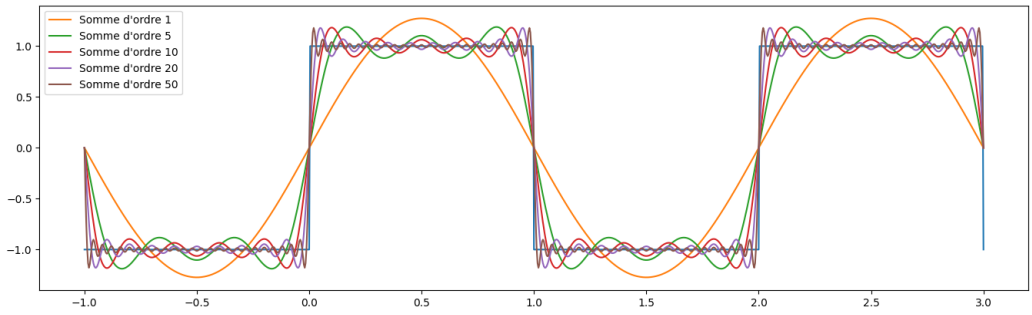
Solution de l'exercice XIX.8 -

```
a = -1
b = 3
n = 1000
pas  = (b-a)/n
X = [a + k*pas for k in range(n+1)]
Y = [rectangle(x) for x in X]
plt.plot(X, Y)

A = liste_a(rectangle, 50, 2)
B = liste_b(rectangle, 50, 2)

for N in [1, 5, 10, 20, 50]:
    Y = [somme_Fourier(A[ : N+1], B[ : N+1], 2, x) for x in X]
    plt.plot(X, Y, label="Somme d'ordre {}".format(N))
plt.legend()
plt.show()
```

Comme la fonction est impaire les coefficients a_n sont tous nuls.



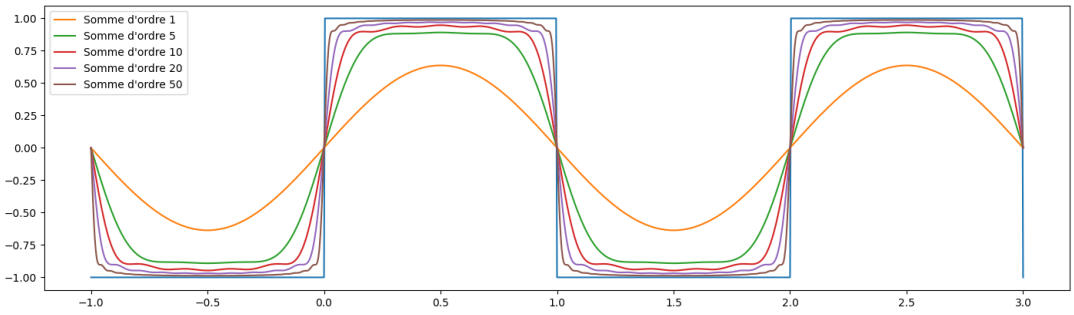
Solution de l'exercice XIX.9 -

```
def somme_Fejer(A, B, T, x):
    N = len(A)
    s = 0
    w = 2*pi/T
    for n in range(N):
        s = s + (A[n]*cos(n*w*x) + B[n]*sin(n*w*x))*(1-n/N)
    return s
```

```
a = -1
b = 3
n = 1000
pas  = (b-a)/n
X = [a + k*pas for k in range(n+1)]
Y = [rectangle(x) for x in X]
plt.plot(X, Y)

A = liste_a(rectangle, 50, 2)
B = liste_b(rectangle, 50, 2)

for N in [1, 5, 10, 20, 50]:
    Y = [somme_Fejer(A[ : N+1], B[ : N+1], 2, x) for x in X]
    plt.plot(X, Y, label="Somme d'ordre {}".format(N))
plt.legend()
plt.show()
```



TP XX

Méthode de Newton

1 L'algorithme

On rappelle l'algorithme associé à la méthode de Newton

```
def Newton(f, f_prime, a, epsilon):
    """ Entrées : deux fonctions réelles, 2 réels
        Requis : f est C1 sur un intervalle contenant a
                 f_prime est la dérivée de f
                 epsilon > 0
        Sortie : c appartenant à [a;b] tel qu'il existe un
                 zero c' de f avec |c - c'| de l'ordre de
                    epsilon """
    x = a
    ecart = 1 + epsilon
    while ecart >= epsilon:
        x_old = x
        x = x - f(x)/f_prime(x)
        ecart = abs(x - x_old)
    return x
```

Pour visualiser le comportement de cet algorithme, on souhaite suivre les évolutions des valeurs calculées. On remplace alors le critère d'approximation (ε) par un critère de nombre d'itérations.

Exercice XX.1
Écrire une fonction `suite_Newton(f, df, x0, n)` *renvoyant la liste des valeurs de la suite* (x_k) *définie par la méthode de Newton appliquée à* f*, de premier terme* x_0*, pour* $0 \leqslant k \leqslant n$.

2 Un exemple

On considère le polynôme $P = X^4 + X^3 - 23X^2 + 3X + 90$

Exercice XX.2
Créer les fonctions `P(x)` *et* `dP(x)` *qui calculent respectivement la valeur de* $P(x)$ *et de* $P'(x)$.

On va rechercher des racines de P en utilisant la méthode de Newton.

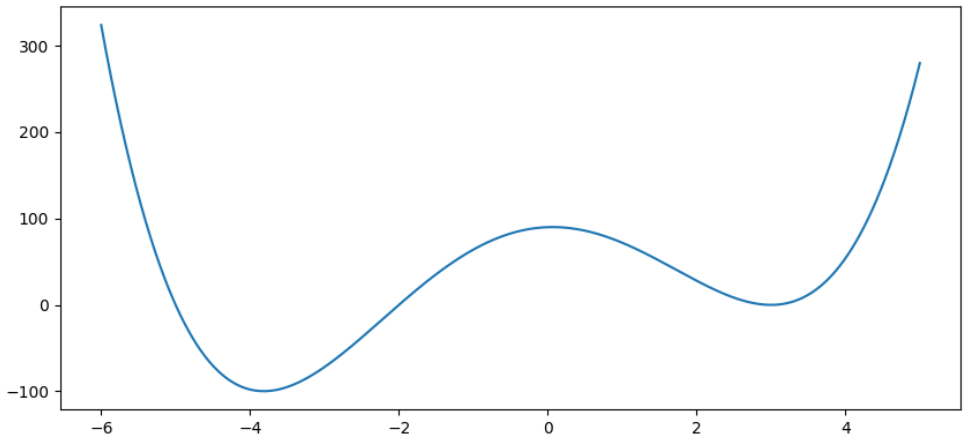


Figure XX.1 – Graphe du polynôme P

Exercice XX.3
Déterminer les suites de valeurs fournies par `suite_Newton(p, dP, x0, 10)` *avec* x_0 *prenant les valeurs dans* $\{-10, -8, -6, -4, -2, 0, 2, 4\}$.
Tracer sur un seul graphe les lignes formées des points (k, x_k) *pour chaque valeur initiale.*
Quelles semblent être les racines de P *?*

3 Sensibilité aux conditions initiales

Dans les calculs précédents on a vu que les zéros calculés pouvaient s'entremêler : les conditions initiales -4, -2 et 0 donnaient respectivement -5, -2 et 0 comme valeur de racines de P. Nous allons explorer ce genre de phénomène.

Exercice XX.4
Écrire une fonction `suite(a, b, n)` *qui renvoie une liste de* n *valeurs régulièrement espacées entre* a *et* b.

Exercice XX.5
Après avoir déterminé `X = suite(-5, 2, 100000)` *calculer la liste* `Y` *telle que* `Y[i]` *est le zéro renvoyé par* `Newton(p, dP, X[i], 0.00001)` *et tracer le graphe correspondant (*`plt.plot(X, Y)`*).*

Si on agrandit le graphe autour de $x = -0.5361498906$ on voit apparaître un comportement fractal : le même motif apparaît à plusieurs échelles.

4 Sans la dérivée

Il est parfois difficile de connaître la dérivée d'une fonction.

- Dans ce cas on peut utiliser la méthode de dichotomie, mais elle est moins efficace que la méthode de Newton.
- On peut aussi utiliser, dans la méthode de Newton, la droite passant par $\big(x, f(x)\big)$ et $\big(x_{old}, f(x_{old})\big)$ (la sécante) à la place de la dérivée. x et x_{old} sont les deux derniers points calculés par la méthode.
  On voit qu'on a besoin de deux points initiaux pour démarrer l'algorithme.
  L'équation de la sécante est $Y = f(x) + \dfrac{X - x}{x_{old} - x}\big(f(x_{old}) - f(x)\big)$ donc le point d'intersection de la sécante avec l'axe des abscisses est $(x', 0)$ avec $x' = x - \dfrac{f(x)(x - x_{old})}{f(x) - f(x_{old})}$.

Exercice XX.6 — Méthode de la sécante
Écrire une fonction `secante(f, a, b, epsilon)` *retournant une valeur approchée à ε près d'une racine r de f en utilisant la méthode de la sécante ; a et b sont les deux valeurs utilisées pour la première sécante.*

- Une autre modification de la méthode de Newton est de remplacer le calcul de la dérivée par une valeur approchée.
  On choisit l'approximation symétrique : $f'(x) \sim \dfrac{f(x+h) - f(x-h)}{2h}$.

Exercice XX.7
Écrire une fonction `der(f, x, h)` *qui renvoie une valeur approchée de $f'(x)$ par cette méthode.*

Exercice XX.8 — Méthode de la pseudo-dérivée
Écrire une fonction `Newton1(f, a, epsilon)` *retournant une valeur approchée à ε près d'une racine r de f en utilisant la méthode ci-dessus. On pourra choisir $h = \frac{\varepsilon}{10}$.*

- On peut surtout utiliser une fonction de la bibliothèque numérique : `fsolve` se trouve dans la sous-bibliothèque `scipy.optimize`.
  C'est la méthode à privilégier dans les calculs scientifiques.

Exercice XX.9
Lire la documentation de `fsolve`.
Comment calculer un zéro de $x \mapsto a^x - x^2 - 2$ en fonction du paramètre a ?

5 Solutions

Solution de l'exercice XX.1 -

```
def suite_Newton(f, df, x0, n):
    X = [0]*(n+1)
    X[0] = x0
    for i in range(n):
        x = X[i]
        X[i+1] = x - f(x)/df(x)
    return X
```

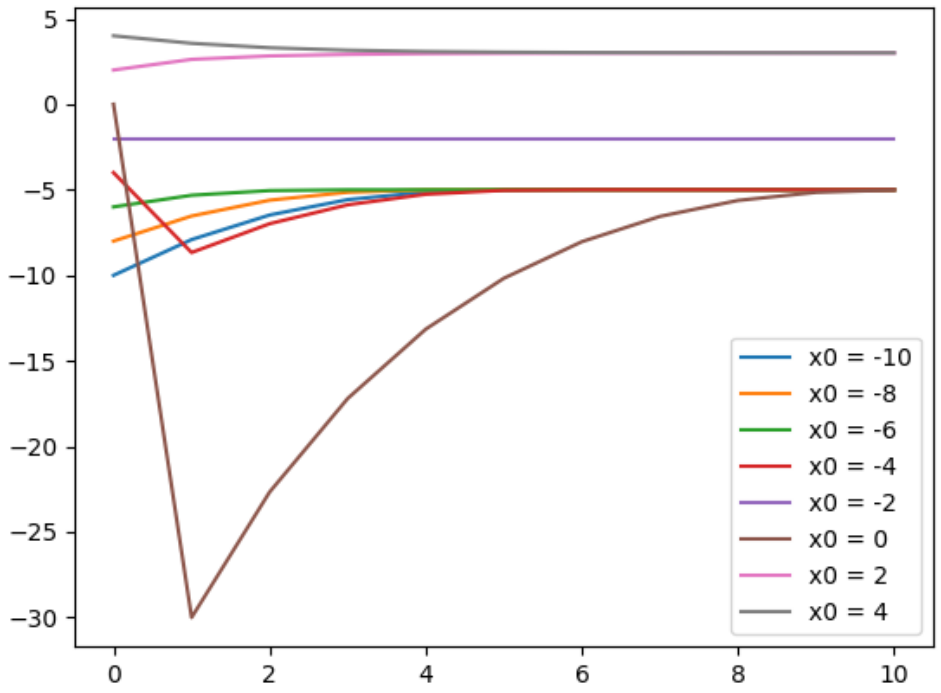
Solution de l'exercice XX.2 -

```
def P(x):
 return 90 + x*(3 + x*(-23+x*(1+x)))

def dP(x):
 return 3 + x*(-46+x*(3+x*4))
```

Solution de l'exercice XX.3 -

```
for x0 in [-10, -8, -6, -4, -2, 0, 2, 4]:
    X = suite_Newton(P, dP, x0, 10)
    plt.plot(X)
plt.show()
```

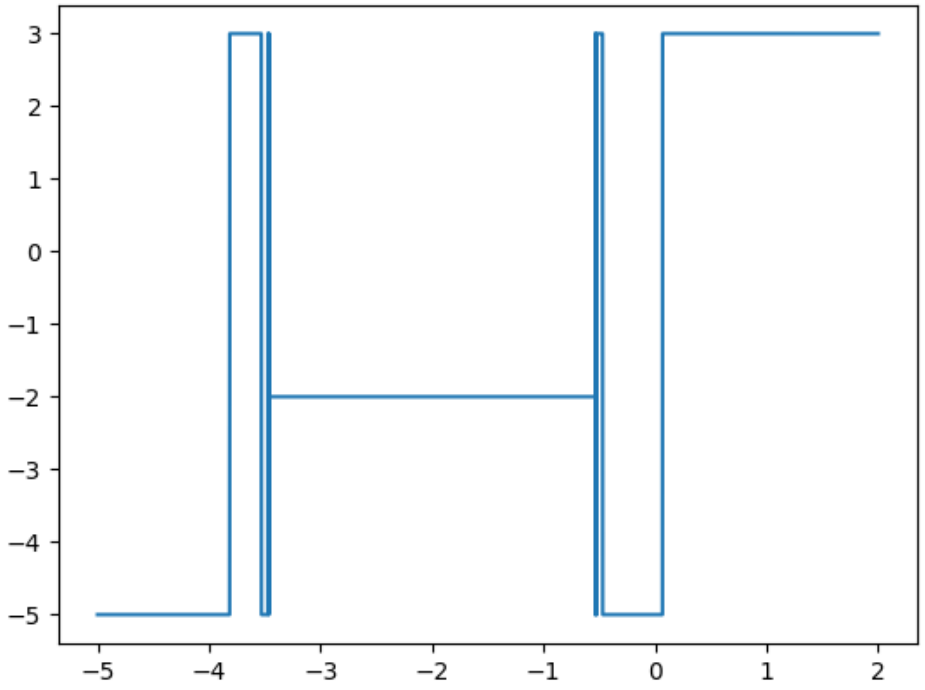


Solution de l'exercice XX.4 -

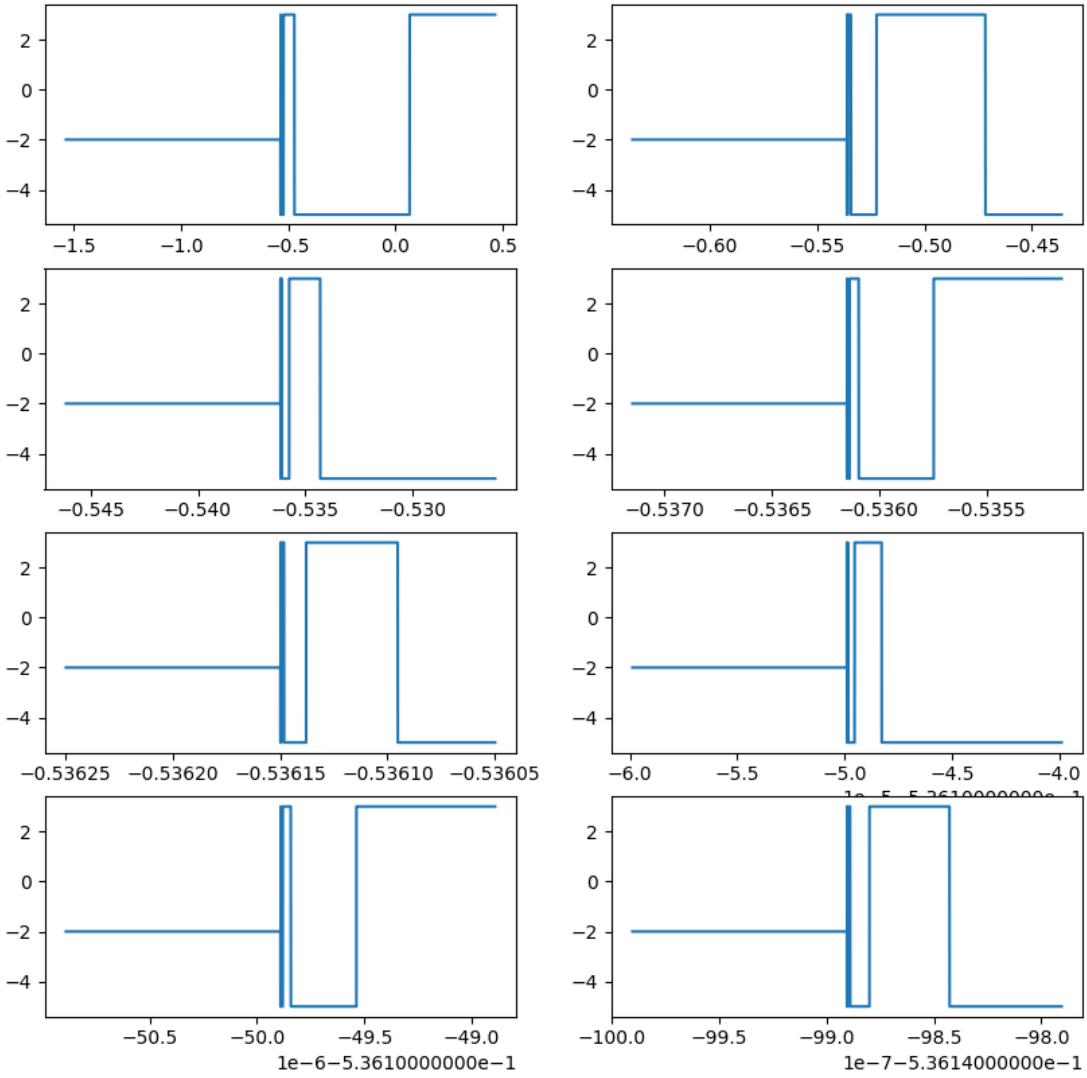
```
def suite(a, b, n):
    pas  = (b-a)/(n-1)
    X = [0]*n
    for i in range(n):
        X[i] = a + pas*i
    return X
```

Solution de l'exercice XX.5 -

```
X = suite(-5, 2, 100000)
Y = [Newton(P, dP, x, 0.0001)[0] for x in X]
plt.plot(X, Y)
plt.show()
```




```
a = -0.5361498906
n = 8
for k in range(n):
    h = 10**(-k)
    X = suite(a-h, a+h, 10000)
    Y = [Newton(P, dP, x, 0.0001) for x in X]
    plt.subplot((n+1)//2, 2, k+1)
    plt.plot(X, Y)
plt.show()
```



Solution de l'exercice XX.6 -

```
def secante(f, a, b, epsilon):
    x_old = a
    x = b
    while abs(x - x_old) >= epsilon:
        x_new = x - f(x)*(x -x_old)/(f(x) - f(x_old))
        x_old = x
        x = x_new
    return x
```

Solution de l'exercice XX.7 -

```
def der(f, x, h):
    return (f(x+h) - f(x-h))/(2*h)
```

Solution de l'exercice XX.8 -

```
def Newton1(f, a, epsilon):
    h = epsilon/10
    x = a
    ecart = 1 + epsilon
    while ecart >= epsilon:
        x_old = x
        x = x - f(x)/der(f, x, h)
        ecart = abs(x - x_old)
    return x
```

Solution de l'exercice XX.9 -

```
from scipy.optimize import fsolve

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

z = fsolve(f, 3, args = (2))
```

TP XXI

Méthode d'Euler

1 Méthode d'Euler explicite

Programme XXI.1 – Euler explicite

```
1  def Euler(phi, y0, T):
2      """Entree : une fonction f de 2 variables
3                  un reel y0, la condition initiale en t0
4                  une liste de réels, monotone, dont le premier
                       terme est t0
5         Sortie : une liste de  points définissant
6                    une solution approchee de y' = phi(y, t)
7                    avec les conditions initiales (t0,y0)
8                    y(T[i]) est approché par Y[i]"""
9      n = len(T)            # n valeurs à calculer
10     Y = [0]*n             # création de la liste
11     Y[0] = y0             # ordonnee initiale
12     for k in range(n-1): # il reste (n-1) points à trouver
13         pas = T[k+1] - T[k]
14         pente = phi(Y[k],T[k])
15         Y[k+1] = Y[k] + pas*pente # On applique la formule
16     return Y
```

1.1 Peu de points

On commence par illustrer la méthode d'Euler sur un exemple pour lequel on connaît la solution. Pour mettre en évidence la convergence des calculs on ne considérera pas un grand nombre de points. Cette situation est l'opposée de l'utilisation usuelle de la méthode : si on veut approcher une solution qu'on ne sait pas calculer on choisit un grand nombre de points.
On considère l'équation $y' = y + t$ avec $y(0) = -0.5$ dont la solution est $t \mapsto \frac{1}{2}e^t - t - 1$.

Exercice XXI.1
Écrire la fonction `phi1(y, t)` *qui renvoie* $y + t$
et la fonction $f(t)$ *qui renvoie* $\frac{1}{2}e^t - t - 1$

Exercice XXI.2
Compléter les instructions suivantes pour qu'elles calculent les solutions calculées sur $[0;3]$ *de l'équation pour des listes de* n *points avec* $n \in \{3, 6, 10, 20, 40, 100\}$ *et les affichent sur un même graphe puis calcule les listes avec 10000 points qui permettent de tracer le graphe de la solution, sur le même graphe.*

Programme XXI.2 – Tracés multiples

```
liste_n = [3, ............]
p = len(liste_n)
for n in liste_n:
    T = np.linspace(......)
    Y = ...........   # solution approchée avec Euler
    plt.plot(T, Y, label = "Euler avec {} points".format(n)

T = ...........
Y = sol(T)  # solution exacte
plt.plot(T, Y, label = "Solution exacte")

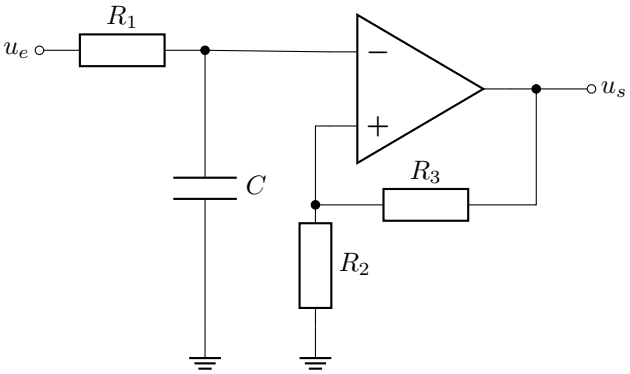
plt.legend()
plt.show()
```

1.2 Étude d'un filtre actif passe bas

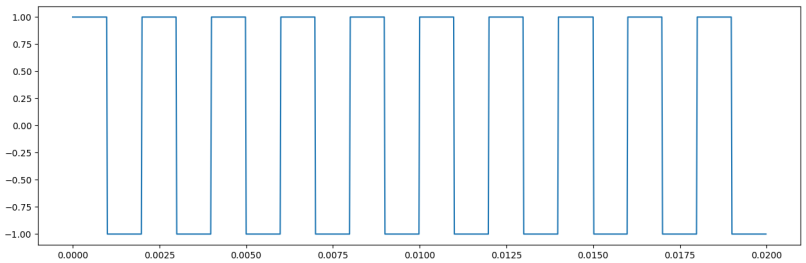
On peut réaliser un filtre actif passe-bas avec un A.O. (Amplificateur opérationnel).
Le fonctionnement de l'amplificateur opérationnel implique que la tension de sortie, u_s, est solution de

$$R_1 C y'(t) + y(t) = \frac{R_2 + R_3}{R_2} u_e(t)$$

où u_e est la tension d'entrée.



Les valeurs numériques sont : $R_2 = R_3 = 1000\Omega$ et $C = 1\mu$F.
Différentes valeurs de R_1 seront testées afin de mettre en évidence les différents comportements.
Le signal d'entrée $u_e(t)$ est un signal rectangulaire de fréquence 500 Hz.



Exercice XXI.3
Écrire une fonction `creneau(t,T)` *qui calcule la valeur d'un signal rectangulaire de période* T *: la valeur devra être 1 sur* $[0; \frac{T}{2}[$ *et* -1 *sur* $[\frac{T}{2}; T[$.

On pourra utiliser l'expression `t%T` qui renvoie le réel x appartenant à $[0;T[$ tel que $t - x$ est un multiple entier de T.

Exercice XXI.4
En utilisant la méthode d'Euler déterminer alors les solutions de l'équation : on prendra une liste de temps de 1000 points entre 0 et 0,02s avec une condition initiale nulle.
On calculera les solutions pour $R_1 = 100\Omega$, $R_1 = 200\Omega$, $R_1 = 500\Omega$, $R_1 = 1000\Omega$ *et* $R_1 = 2000\Omega$.
On pourra tracer les solutions en même temps que le signal d'entrée.

2 Méthode d'Euler implicite

La méthode d'Euler implicite utilise la formule

$$y_{k+1} - y_k = (t_{k+1} - t_k)\varphi(y_{k+1}, t_{k+1})$$

Dans cette équation les temps, t_k et t_{k+1}, sont connus ainsi que la valeur (approchée) de la solution en t_k, y_k. Elle demande de résoudre une équation en y_{k+1}.
Elle permet parfois une meilleure stabilité de la solution.

2.1 Exemple simple

On peut illustrer la stabilité avec l'équation $\begin{cases} y' = -20y \\ y(0) = y_0 \end{cases}$.

Exercice XXI.5
Déterminer et tracer la solution approchée de l'équation ci-dessus par la méthode d'Euler explicite sur $[0; 10]$ *avec 100 points pour* $y_0 = 1$.
On pourra aussi regarder le comportement pour 101 points puis 102.

On ne retrouve pas le résultat attendu, la solution est $t \mapsto e^{-20t}$.

Exercice XXI.6
Résoudre (mathématiquement) l'équation d'inconnue y_{k+1},

$$y_{k+1} = y_k + (t_{k+1} - t_k)\varphi(y_{k+1}, t_{k+1}) = y_k - 20(t_{k+1} - t_k)y_{k+1}$$

En déduire une fonction `solution6(y0, T)` *de résolution de l'équation en fonction de* y_0 *et de la liste des temps puis tracer la solution pour* $y_0 = 1$ *avec 100 points entre 0 et 10.*

2.2 Autre exemple

On considère l'équation $\begin{cases} y' = ty - y^2 \\ y(0) = y_0 \end{cases}$.

Exercice XXI.7
Montrer que la méthode d'Euler implicite donne le calcul

$$y_{k+1} = \frac{ht_{k+1} - 1 \pm \sqrt{(ht_{k+1} - 1)^2 + 4hy_k}}{2h}$$

où h *est le pas de temps* $t_{k+1} - t_k$.
Pourquoi est-il judicieux de choisir la solution $\dfrac{ht_{k+1} - 1 + \sqrt{(ht_{k+1} - 1)^2 + 4hy_k}}{2h}$ *?*

Exercice XXI.8
Déterminer une fonction de résolution de l'équation (E_2) *en fonction de* y_0 *et de la liste des temps :* `solution8(y0, T)`.
Comparer graphiquement les solutions par les deux méthodes (implicite et explicite) sur $[0; 4]$ *avec 50 puis 500 points pour* $y_0 = 10$.

3 Autres méthodes de résolution

3.1 Bibliothèque `scipy`

Dans la bibliothèque `scipy`, on dispose d'une fonction `odeint`

```
from scipy.integrate import odeint
```

La fonction `odeint` est alors substituable, sans modification, à la fonction `Euler`.
Elle donnera des résultats plus précis.

3.2 Schéma de Heun

La méthode de Heun améliore le schéma d'Euler en approchant l'intégrale par la méthode des trapèzes : $\displaystyle\int_a^b g(u)\mathrm{d}u \simeq (b-a)\frac{g(a)+g(b)}{2}$. On obtient

$$y(t_{k+1}) - y(t_k) \simeq (t_{k+1}-t_k)\frac{\varphi\big(y(t_k),t_k\big)+\varphi\big(y(t_{k+1}),t_{k+1}\big)}{2}$$

Cependant cela devient une équation implicite en $y(t_{k+1})$ qu'on ne souhaite pas résoudre.
On va donc procéder en approchant la valeur de $y(t_{k+1})$ dans le second membre par celle que calcule la méthode d'Euler.

- On calcule $z_k = y_k + \varphi(y_k,t_k)(t_{k+1}-t_k)$ comme valeur approchée de $y(t_{k+1})$
- On calcule la valeur approchée $y_{k+1} = y_k + \dfrac{\varphi(y_k,t_k)+\varphi(z_k,t_{k+1})}{2}(t_{k+1}-t_k)$.

On remarque que z_k est la valeur approchée par la méthode d'Euler.

Exercice XXI.9
Écrire une fonction `Heun(phi, y0, T)` *qui approche, aux points de* `T`*, une solution de l'équation* $y' = \varphi(y,t)$ *avec la condition initiale* $y(t_0) = y_0$ *où* t_0 *est la première valeur de la liste* `T`*. Le docstring est le même que celui de la fonction* `Euler`*.*

3.3 Méthode de Runge-Kutta

La méthode de Runge-Kutta s'inspire de la méthode de Simpson :

$$\int_a^b g(u)\mathrm{d}u \simeq (b-a)\frac{g(a)+4g\big(\frac{a+b}{2}\big)+g(b)}{6}$$

On obtient, pour $m_k = \frac{t_k+t_{k+1}}{2}$ et $h = t_{k+1}-t_k$,

$$y(t_{k+1}) \simeq y(t_k) + h\frac{\varphi\big(y(t_k),t_k\big)+4\varphi\big(y(m_k),m_k\big)+\varphi\big(y(t_{k+1}),t_{k+1}\big)}{6}$$

On va, ici encore, approcher provisoirement les valeurs de $y(m_k)$ et $y(t_{k+1})$.
On va en fait utiliser deux approximations de $y(m_k)$.

- $a = y_k + \frac{h}{2}\varphi\big(y_k,t_k\big)$ est une première valeur approchée de $y(m_k)$
- $b = y_k + \frac{h}{2}\varphi\big(a,t_k+\frac{h}{2}\big)$ est aussi une valeur approchée de $y(m_k)$
- $c = y_k + h\varphi\big(b,t_k+\frac{h}{2}\big)$ est une valeur approchée de $y(t_{k+1})$ obtenue en prenant comme valeur moyenne la valeur au point milieu.
- On pose alors
  $y_{k+1} = y_k + h\dfrac{\varphi\big(y_k,t_k\big)+2\varphi\big(a,t_k+\frac{h}{2}\big)+2\varphi\big(b,t_k+\frac{h}{2}\big)+\varphi\big(c,t_k+h\big)}{6}$

Exercice XXI.10
Écrire une fonction `RK(phi, y0, T)` *avec le même docstring que celui de la fonction* `Euler`*.*

3.4 Méthode d'Euler implicite

On peut écrire une méthode de résolution générale avec la méthode d'Euler implicite en définissant une fonction à résoudre à chaque étape et en utilisant une méthode de résolution ; par exemple `fsolve` qui reçoit comme paramètres une fonction et un point de départ. Pour calculer `Y[k+1]` en prenant `Y[k]` comme point de départ.

```
from scipy.optimize import fsolve
```

Exercice XXI.11
Écrire une fonction `Euler_implicite(phi, y0, T)` *avec le même docstring que celui de la fonction* `Euler`.

3.5 Comparaisons

On va comparer les différents schémas.
On considère l'équation $\begin{cases} y' = y - 2e^{-t} \\ y(0) = 1 \end{cases}$.
La solution est $t \mapsto e^{-t}$; on va l'approcher sur $[0; 30]$ avec 10000 points.
On verra que chaque méthode donne un résultat qui finit par diverger.

Exercice XXI.12
Calculer et représenter les solutions pour les différentes méthodes.

4 Solutions

Solution de l'exercice XXI.1 -

```
def phi1(y, t):
    return y + t
```

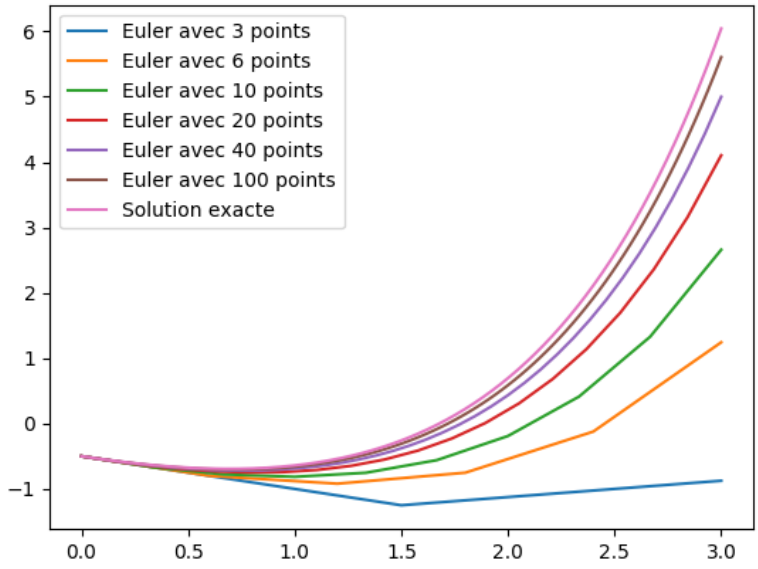
```
def sol(t):
    return np.exp(t)/2 - t - 1
```

Solution de l'exercice XXI.2 -

```
liste_n = [3, 6, 10, 20, 40, 100]
for n in liste_n:
    T = np.linspace(0, 3, n)
    Y = Euler(phi1, -0.5, T)
    plt.plot(T, Y, label = "Euler avec {} points".format(n))

T = np.linspace(0, 3, 10000)
Y = sol(T)
plt.plot(T, Y, label = "Solution exacte")

plt.legend()
plt.show()
```



Solution de l'exercice XXI.3 -

```
def creneau(t, T):
    x = t%T
    if x < T/2:
        return 1
    else:
        return -1
```

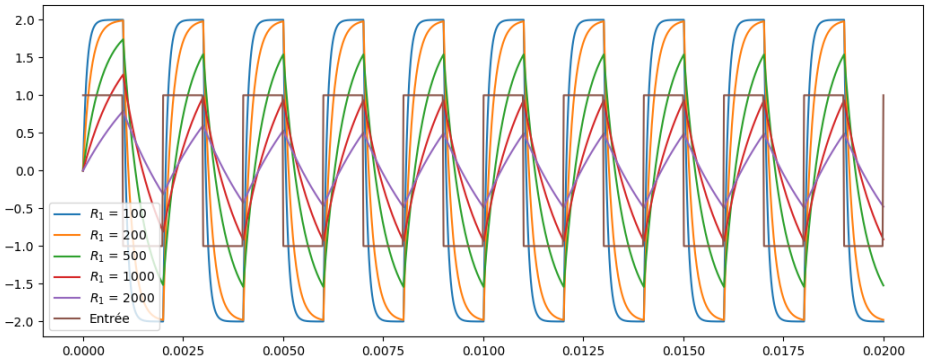
Solution de l'exercice XXI.4 -

```
def filtre(y, t):
    return ((R2 + R3)/R2*creneau(t, per) - y)/R1/C
```

```
R2 = 1000
R3 = 1000
f = 500
per = 1/f
C = 1e-6
R = [100, 200, 500, 1000, 2000]
T = np.linspace(0, 0.02, 1000)
for R1 in R:
    Y = Euler(filtre, 0, T)
    plt.plot(T, Y, label = "$R_1$ = {}".format(R1))

def cr(t):
    return creneau(t, per)

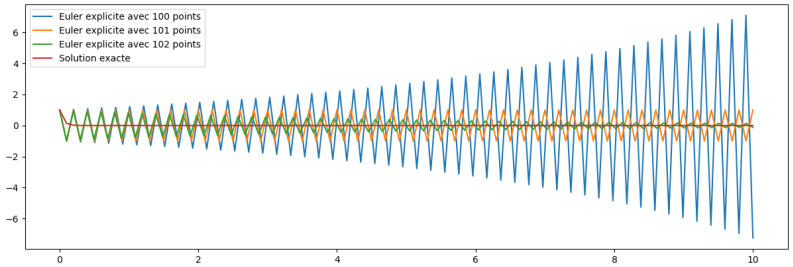
f_cr = np.vectorize(cr)
Y = f_cr(T)
plt.plot(T, Y, label = "Entrée")
plt.legend()
plt.show()
```



Solution de l'exercice XXI.5 -

```
def phi5(y, t):
    return -20*y
```

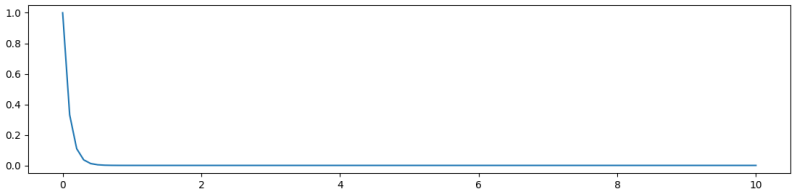
```
for n in range(100, 103):
    T = np.linspace(0, 10, n)
    Y = Euler(phi5, 1, T)
    plt.plot(T, Y, label = "Euler explicite avec {} points".
        format(n))
Y = np.exp(-20*T)
plt.plot(T, Y, label = "Solution exacte")
plt.legend()
plt.show()
```



Solution de l'exercice XXI.6 - L'équation donne $y_{k+1} = \dfrac{y_k}{1 + 20(t_{k+1} - t_k)}$.

```
def solution6(y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for k in range(n-1):
        pas = T[k+1] - T[k]
        Y[k+1] = Y[k]/(1+20*pas)
    return Y
```

```
T = np.linspace(0, 10, 100)
Yi = solution1(1, T)
plt.plot(T, Yi)
plt.show()
```



Solution de l'exercice XXI.7 -
L'équation devient $y_{k+1} - y_k = h.\left(t_{k+1}y_{k+1} - y_{k+1}^2\right)$ donc y_{k+1} est solution de $hY^2 - (ht_{k+1} - 1)Y - y_k = 0$ d'où la formule.

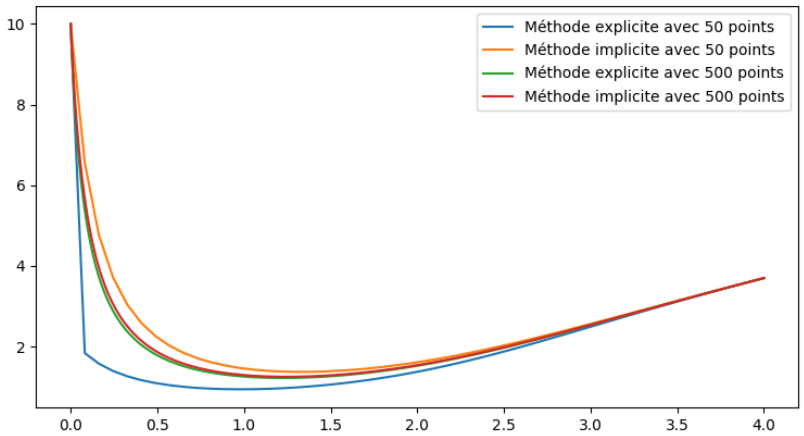
Si on fait un développement limité à l'ordre 1 en h on trouve
$y_{k+1} = \dfrac{ht_{k+1} - 1 \pm \big(1 + 2hy_k - ht_{k+1} + o(h)\big)}{2h}$.
Seul le signe + donne une limite finie (y_k) quand h tend vers 0.

Solution de l'exercice XXI.8 -

```
def solution8(y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for k in range(n-1):
        pas = T[k+1] - T[k]
        delta = (pas*T[k+1] -1)**2 + 4*pas*Y[k]
        Y[k+1] = (pas*T[k+1] - 1 + delta**0.5)/(2*pas)
    return Y
```

```
def phi8(y, t):
    return t*y - y**2
```

```
for n in [50, 500]:
    T = liste_abs(0, 4, n)
    Ye = Euler(phi8, 10, T)
    Yi = solution8(10, T)
    plt.plot(T, Ye,
             label = "Méthode explicite avec {} points".format
                (n))
    plt.plot(T, Yi,
             label = "Méthode implicite avec {} points".format
                (n))
plt.legend()
plt.show()
```



Solution de l'exercice XXI.9 -

```
def Heun(phi, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for k in range(n-1):
        pas = T[k+1] - T[k]
        pente1 = phi(Y[k],T[k])
        z = Y[k] + pas*pente1
        pente2 = phi(z, T[k+1])
        pente = (pente1 + pente2)/2
        Y[k+1] = Y[k] + pas*pente
    return Y
```

Solution de l'exercice XXI.10 -

```
def RK(f, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0] = y0
    for i in range(n-1):
        y = Y[i]
        t = T[i]
        t1 = T[i+1]
        m = (t + t1)/2
        pas = t1 - t
        pente1 = f(y,t)
        a = y + pente1*pas/2 # 1ère valeur au milieu
        pente2 = f(a, m) # 1ère pente au milieu
        b = y + pente2*pas/2 # 2ème valeur au milieu
        pente3 = f(b, m) # 2ème pente au milieu
        c = y + pente3*pas  # valeur à droite
        pente4 = f(c,t1)  # pente à droite
        pente = (pente1 + 2*pente2 + 2*pente3 + pente4)/6
        Y[i+1] = y + pas*pente
    return Y
```

Solution de l'exercice XXI.11 -

```
def Euler_implicite(phi, y0, T):
    n = len(T)
    Y = [0]*n
    Y[0]= y0
    for k in range(n-1):
        pas = T[k+1] - T[k]
        def fk(y):
            return y - Y[k] - pas*phi(y, T[k+1])
        Y[k+1] = fsolve(fk, Y[k])
    return Y
```

Solution de l'exercice XXI.12 -

```
def phi12(y, t):
    return y -2*np.exp(-t)

y0 = 1
T = np.linspace(0, 30, 10000)

Y = Euler(phi12, y0, T)
plt.plot(T, Y, label = "Euler explicite")

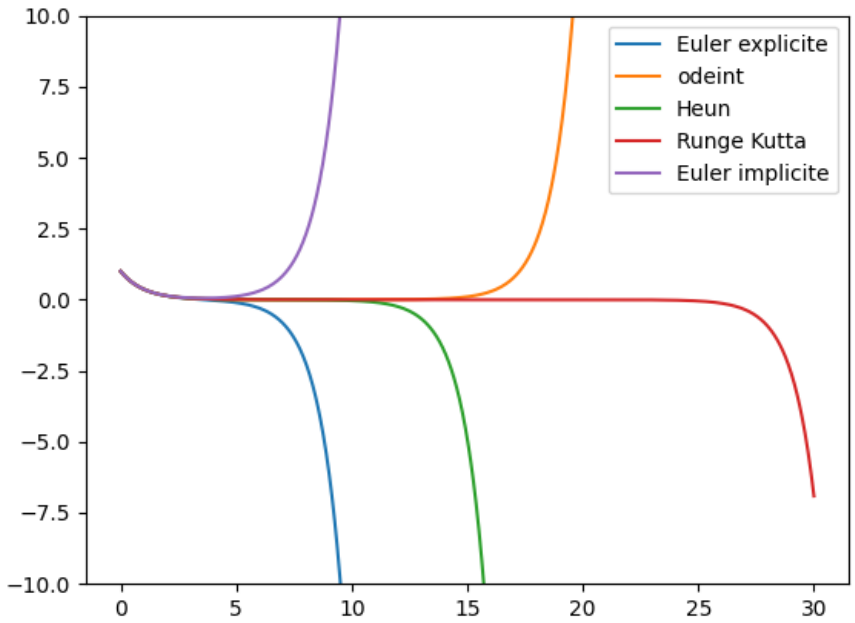
Y = odeint(phi12, y0, T)
plt.plot(T, Y, label = "odeint")

Y = Heun(phi12, y0, T)
plt.plot(T, Y, label = "Heun")

Y = RK(phi12, y0, T)
plt.plot(T, Y, label = "Runge Kutta")

Y = Euler_implicite(phi12, y0, T)
plt.plot(T, Y, label = "Euler implicite")

plt.ylim([-10, 10])
plt.legend()
plt.show()
```



TP XXII

Euler vectoriel

Dans ce T.P. nous allons utiliser les tableaux `numpy` comme des vecteurs, nous allons tracer des représentations graphiques de solutions ou de portraits de phase et nous pourrons utiliser la méthode de résolution `odeint`.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint
```

1 Systèmes différentiels

Un système différentiel $\begin{cases} x' = \phi_1(x, y, \dots, t) \\ y' = \phi_2(x, y, \dots, t) \\ \qquad \dots \end{cases}$ est considéré comme une équation différentielle d'ordre 1 dont la fonction à rechercher est une fonction à valeurs vectorielles. On écrit la fonction définissant l'équation différentielle sous la forme

```
def phi(u, t):
    x, y, ... = u
    dx = # calcul de phi1(x, y, ..., t)
    dy = # calcul de phi2(x, y, ..., t)
    ...
    return np.array([dx, dy, ...])
```

La condition initiale sera aussi un tableau `numpy`

```
u0 = np.array([x0, y0, ...])
```

La liste des temps peut être calculée par la fonction `numpy`

```
T = np.linspace(t0, tFin, N)
```

On peut alors invoquer toutes les méthodes vues, le plus souvent `Euler` ou `odeint`

```
U = Euler(phi, u0, T)
```

`U` est alors une liste de vecteurs dont on peut extraire les composantes

```
X = [u[0] for u in U]
Y = [u[1] for u in U]
...
```

1.1 Proies-prédateurs

Un modèle d'évolution de deux espèces a été proposé dans les années 1920 par Alfred Lotka (1880-1949) et Vito Volterra (1860-1940). Il modélise les évolutions de deux type d'animaux, les **proies** et les **prédateurs** dont les population sont représentées respectivement par les variables p et q **à valeurs réelles**.

- Les proies disposent de nourriture en quantité suffisante, en l'absence de prédateurs l'évolution est régie par un taux de natalité α : $\dfrac{\mathrm{d}p}{\mathrm{d}t} = \alpha p$.
- Les prédateurs n'ont pas d'autre nourriture que les proies, en l'absence de celles-ci elles meurent avec un taux β : $\dfrac{\mathrm{d}q}{\mathrm{d}t} = -\beta q$.
- Les prédateurs peuvent croître (et les proies décroître) lors des rencontres dont le nombre est estimé proportionnel au produit pq des populations.

$$\text{On parvient donc aux lois d'évolution } \begin{cases} \dfrac{\mathrm{d}p}{\mathrm{d}t} = \alpha p - \gamma pq \\ \dfrac{\mathrm{d}q}{\mathrm{d}t} = -\beta q + \delta pq \end{cases}$$

On donne les valeurs des coefficients dans des variables globales. Dans une étude théorique, on peut simplifier les valeurs.

```
alpha = 1
beta  = 2
gamma = 0.001
delta = 0.001
```

Exercice XXII.1
Définir les fonctions de l'équation différentielle.
Résoudre cette équation sur $[0;20]$ *avec 10000 points pour les conditions initiales* $p0 = 6000$ *et* $q_0 = 1000$.
Tracer les solutions $p(t)$ *et* $q(t)$ *en fonction de* t *puis la trajectoire* $\big(p(t), q(t)\big)$.

On remarque que, si on a utilisé la méthode d'Euler, la trajectoire ne se referme pas. On peut prouver que les solutions décrivent une courbe fermée. Dans la suite on utilisera `odeint`.

Exercice XXII.2
Résoudre l'équation sur $[0;20]$ *avec 10000 points pour les conditions initiales* $p0 \in \{2200, 2500, 3000, 4000, 6000, 10000\}$ *et* $q_0 = 1000$ *et tracer, sur un même graphe les trajectoires.*

1.2 Simulation d'une épidémie

On modélise la propagation d'une épidémie en séparant la population en 3 :

- les individus **sains** dont la proportion est représentée par une fonction $S(t)$,
- les personnes **infectées** de proportion $I(t)$ et
- les guéris comptés par $R(t)$ (pour *recovered*).

On parle du modèle SIR.
On définit deux constantes β et γ qui représente respectivement le taux de transmission, c'est à dire le taux de personnes saines qui deviennent infectées par contact avec les personnes infectées et le taux de guérison, c'est à dire le taux de personnes infectées qui deviennent guéries.
On suppose que les personnes guéries ne peuvent plus être infectées.
Mathématiquement, le modèle SIR est donné par le système :

$$\begin{cases} \dfrac{\mathrm{d}S}{\mathrm{d}t}(t) &=& -\beta S(t)I(t) & & \\ \dfrac{\mathrm{d}I}{\mathrm{d}t}(t) &=& \beta S(t)I(t) & - & \gamma I(t) \\ \dfrac{\mathrm{d}R}{\mathrm{d}t}(t) &=& & & \gamma I(t) \end{cases}$$

On note R_0 le rapport $\dfrac{\beta}{\gamma}$, il mesure la propagabilité de la maladie.
On prendra comme valeurs $R_0 = 3$ et $\gamma = 0.2$.
On considérera qu'on a $I(0) = 10^{-3}$ et $R(0) = 0$ et on étudiera l'évoilution pour $t \in [0; 100]$

Exercice XXII.3
Tracer, sur un même graphique, les proportions de chaque type de population pour t variant entre 0 et 100.

On ajoute une période de confinement entre les temps $t = 10$ et $t = 40$, pendant laquelle le coefficient $R0$ est divisé par 2, γ restant constant.

Exercice XXII.4
Après avoir défini une nouvelle fonction ϕ, tracer, sur un même graphique, les proportions de chaque type de population.

1.3 Équations de Lorenz

Lors de l'étude des phénomènes météorologiques, Edward Lorenz a simplifié, en 1963, les équations issues de la mécanique des fluides en un système dynamique tridimensionnel. Ce système engendre un comportement chaotique dans certaines conditions. Il est donné par

$$\begin{cases} x' &= \sigma(y-x) \\ y' &= \rho x - y - xz \\ z' &= xy - \beta z \end{cases}$$

avec $\sigma = 10$, $\rho = 28$ et $\beta = 8/3$. On peut tracer une trajectoire dans l'espace avec la fonction `Axes3D`.

```
from mpl_toolkits.mplot3d import Axes3D
```

Le tracé 3D est fait en définissant un cadre ; `mon_dessin` est le nom choisi.
Pour les version de `numpy` jusqu'à 3.3

```
fig = plt.figure()
mon_dessin = Axes3D(fig)
```

Pour les version de `numpy` à partir de 3.4

```
fig = plt.figure()
mon_dessin = Axes3D(fig, auto_add_to_figure = False)
fig.add_axes(mon_dessin)
```

On peut alors tracer la trajectoire, définie par 3 listes `X`, `Y` et `Z` de même longueur.

```
mon_dessin.plot(X, Y, Z)
plt.show()
```

Exercice XXII.5
Tracer le portrait de phase sur $[0; 30]$ pour les conditions initiales $x(0) = 0.1$, $y(0) = 0$ et $z(0) = 0.1$.

2 Équations d'ordre 2

On rappelle que, pour résoudre ces équations, on introduit une variable supplémentaire, la dérivée de y, notée ici v. L'équation $y'' = \psi(y, y', t)$ devient alors le système

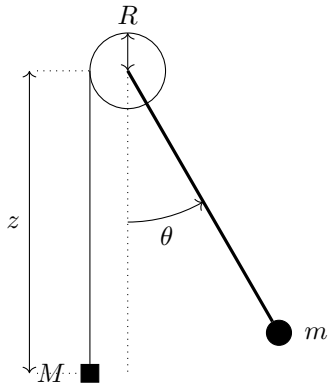
$$\begin{cases} y' = v \\ v' = \psi(y, v, t) \end{cases}$$

2.1 Un exemple simple avec erreur d'arrondi

Exercice XXII.6
Tracer les solutions de $y'' = 2y$ *avec* $y(0) = 1$ *et* $y'(0) = -\sqrt{2}$ *sur* $[0; 30]$.

2.2 Pendule asymétrique

Une masse m est fixée sur une tige rigide de longueur l, solidaire d'une poulie de rayon R. Un fil est enroulé autour de la poulie, une masse M étant fixée à son extrémité libre. Le fil étant inextensible, $z = R\theta$. Le problème est ainsi ramené à un système à un seul degré de liberté θ. Son énergie mécanique E_m est une constante qui s'exprime à tout instant par :



$$E_m = \frac{1}{2}(ml^2 + MR^2)\dot{\theta}(t)^2 + mgl(1 - \cos\big(\theta(t)\big)) - MgR\theta$$

que l'on ramène par dérivation à :

$$\ddot{\theta}(t) + \frac{mgl\sin\big(\theta(t)\big) - MgR}{ml^2 + MR^2} = 0$$

Si M est trop lourde, m ne pourra pas l'empêche de chuter. Alors la tige va entrer en révolution autour de la poulie : θ sera non bornée. Idem si l'une des deux masses se voit imprimer une vitesse initiale trop importante. Sinon, les deux masses oscilleront, l'une angulairement et l'autre linéairement, autour d'une position d'équilibre.

```
R = 0.05   #m
l = 0.5    #m
m = 0.1    #kg
g = 9.81   #m.s^{-2}
```

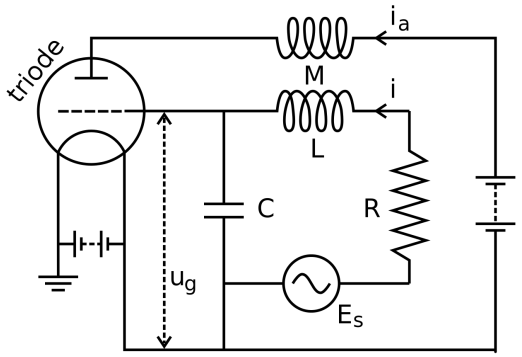
Exercice XXII.7
Tracer les solutions pour le temps variant entre 0 et 5s, pour M prenant les valeurs : 0,65kg, 0,70 kg, 0.72 kg, 0.73 kg et pour les conditions initiales : $\theta(0) = 0$ *et* $\dot{\theta}(0) = 0$.

Exercice XXII.8
Tracer, sur un même graphique, les portraits de phases on se restreindra à l'intervalle $[0; 2, 5]$ *pour le temps*

2.3 Équation de van der Pol

Ce qui suit est tiré de Wikipedia

L'oscillateur de Van der Pol a été imaginé par le physicien néerlandais Balthasar van der Pol alors qu'il était employé par les laboratoires PHILIPS. Van der Pol découvrit que le circuit ci-contre contenant un tube à vide développait des oscillations stables, qu'il appela *oscillation de relaxation* et que l'on désigne aujourd'hui plutôt comme des cycles limites des circuits électriques.



L'équation vérifiée par u_g est

$$y'' = \varepsilon\omega_0(1-y^2)y' - \omega_0^2 y$$

Dans la suite on prendra $\varepsilon = \omega_0 = 1$.

```
epsilon = 1
omega0 = 1
```

Exercice XXII.9
Écrire la fonction `phi(y, v, t)` *correspondant à l'équation.*
Tracer le portrait de phase de la solution sur $[0;20]$ *pour différentes conditions initiales :* $y_0 \in \{1,3,5\}$ *et* $v_0 \in \{-3,0,3\}$.

On remarque que les courbes finissent par se ressembler, c'est le cycle limite.

3 Systèmes différentiels d'ordre 2

On peut généraliser et étudier des sustèmes de polusieurs équations différentielles couplées d'ordre 2. On se ramène à un système d'ordre 1 en dédoublant les variables : à chaque variable on lui associe sa dérivée.

3.1 Effet Zeeman

L'électron dans l'atome est classiquement modélisé comme «élastiquement lié» : un point matériel de masse m et de charge électrique e, repéré par sa position $\vec{r}$, et soumis de la part du noyau à une force de rappel élastique, anticolinéaire à $\vec{r}$. Il se comporte alors comme un oscillateur harmonique de pulsation ω_0.
Si l'atome est soumis à un champ magnétique $\vec{B}$ colinéaire à $\vec{u}_z$, il subit une seconde force, d'origine magnétique ; il se produit alors un phénomène appelé effet Zeeman :

- L'électron garde un mouvement oscillant à la pulsation ω_0 le long du champ magnétique.
- Il développe un mouvement bi-harmonique dans le plan perpendiculaire au champ magnétique, aux pulsations $\omega_0 \pm \Omega$, où Ω est une pulsation proportionnelle au champ magnétique.
- L'atome peut alors émettre un rayonnement sur les trois pulsations ω_0, $\omega_0 - \Omega$ et $\omega_0 + \Omega$.

Le mouvement suivant z est un simple oscillateur harmonique découplé des deux autres, donc il ne sera pas étudié.
L'équation différentielle gouvernant le vecteur position est $\vec{r}''(t) + \frac{e}{m}\vec{r}'(t) \wedge \vec{B} + \omega_0^2\vec{r} = \vec{0}$. Elle se traduit en le système $\begin{cases} \ddot{x} + \omega_0^2 x &= -2\omega\dot{y} \\ \ddot{y} + \omega_0^2 y &= 2\omega\dot{x} \end{cases}$ avec $\omega = \frac{eB}{2m}$.

L'équation différentielle portera donc sur 4 variables :

```
def phi(u, t):
    x, y, vx, vy = u
    ax = ...
    ay = ...
    return np.array([vx, vy, ax, ay])
```

On choisit, pour conditions initiales, $\vec{r}(0) = \vec{0}$ et $\vec{r}'(0) = a\omega_0\vec{u}_x$.
Valeurs numériques : [1]

```
m = 9.1e-31     #kg, masse de l'électron
e = 1.6e-19     #C, charge de l'électron
w0 = 4.34e15    #rad/s pulsation propre du rappel du noyau
B = 1000        #T, intensité du champ magnétique

w = (e*B)/(2*m)
a = 5.3e-11
```

Exercice XXII.10
Tracer le graphe des solutions $x(t)$ et $y(t)$ sur $[0; 5.10^{-14}]$.
Tracer le diagramme des phases pour la solution $x(t)$.

3.2 Ceintures de Van Halen

Les ceintures de Van Halen sont des régions autour de la terre dans lesquelles les particules chargées issues du soleil peuvent se retrouver piégées. Le champ magnétique terrestre dévie les particules de leur trajectoire vers la terre en leur faisant suivre les lignes de champ. Nous allons observer les trajectoire des protons dans la première ceinture située approximativement à une distance de 2 rayons terrestres du centre de la terre. On donne les valeurs

```
q0 = 1.6e-19  # C, charge du proton
m0 = 1.67e-27 # kg, masse du proton
```

Le champ magnétique terrestre peut être approché par un dipôle magnétique : il est représenté par son origine O, au centre de la terre, par son moment magnétique $\vec{\mu}$ dont la direction est proche de l'axe de la terre. La valeur numérique estimée est $\|\vec{\mu}\| = 7,7.10^{22}\mathrm{A}\cdot\mathrm{s}^{-2}$.
Les vecteurs dans Python seront représentés par un tableau numpy avec la troisième coordonnée indiquant l'axe de la terre

```
mu = np.array([0.0, 0.0, 7.7e22])
```

Le champ magnétique en un point M est alors, en notant $r = \left\|\overrightarrow{OM}\right\|$ et $\vec{u} = \frac{1}{r}\overrightarrow{OM}$,

$$\overrightarrow{B(M)} = \frac{\mu_0}{4\pi}\frac{3(\vec{\mu}.\vec{u})\vec{u} - \vec{\mu}}{r^3}$$

On a $\mu_0 = 4\pi 10^{-7}\mathrm{kg}\cdot\mathrm{m}\cdot\mathrm{A}^{-2}\cdot\mathrm{s}^{-2}$ et on notera `mu0` la valeur de $\frac{\mu_0}{4\pi}$: `mu0 = 1.0 e-7`.
Le produit scalaire de deux vecteurs représentés par des tableaux `u` et `v` de même taille se calcule par `np.dot(u, v)`, leur produit vectoriel par `np.cross(u, v)`

Exercice XXII.11
Écrire une fonction `B(M)` *qui calcule le champ magnétique en un point m représenté par un tableau* numpy *de taille 3 (il représente aussi le vecteur $\overrightarrow{OM}$).*

1. La valeur numérique du champ magnétique est énormément amplifiée pour les besoins de cet exercice, sans ça le phénomène est difficile à voir sur un graphique.

L'équation différentielle dont on cherche les solution est alors

$$m\frac{\overrightarrow{\mathrm{d}^2M}}{\mathrm{d}t^2} = q\frac{\overrightarrow{\mathrm{d}M}}{\mathrm{d}t} \wedge \overrightarrow{B(M)}$$

Cette équation va se traduire en un système différentiel à 6 variables : les 3 coordonnées de x, y et z, coordonnées de M, et les coordonnées du vecteur vitesse.

Exercice XXII.12
Écrire une fonction `phi(u, t)` *qui décrit le système différentiel, elle reçoit deux variables, un tableau de taille 6 et un flottant pour le temps (qui n'intervient pas) et doit renvoyer un tableau de taille 6.*

Les unités de base pour les conditions initiales sont le rayon de la terre pour la position initiale et une fraction de la vitesse de la lumière pour la vitesse initiale. Par exemple :

```
RT = 6.4e6
c = 3.0e8
u0 = np.array([0, 2*RT, 0, 0, 0.25*c, 0.1*c])
```

Exercice XXII.13
Tracer la courbe intégrale d'une solution sur $[0; 5]$.

On peut souhaiter représenter la terre pour voir la position relative de la trajectoire.

```
r = np.linspace(0, RT, 30)
phi = np.linspace(0, 2*np.pi, 20)
R, Phi = np.meshgrid(r, phi)
X1 = R*np.cos(Phi)
Y1 = R*np.sin(Phi)
Z1 = (RT**2 - R**2)**0.5
ax.plot_wireframe(X1, Y1, Z1, color = "gray")
ax.plot_wireframe(X1, Y1, -Z1, color = "gray")
ax.set_xlim([-d*1.1, d*1.1])
ax.set_ylim([-d*1.1, d*1.1])
ax.set_zlim([-1.2*RT, 1.2*RT])
plt.show()
```

`meshgrid` construit deux tableaux à deux dimension à partir de deux tableaux à une dimension en répétant les valeurs sur les colonnes pour le premier tableau et sur les lignes pour le second.

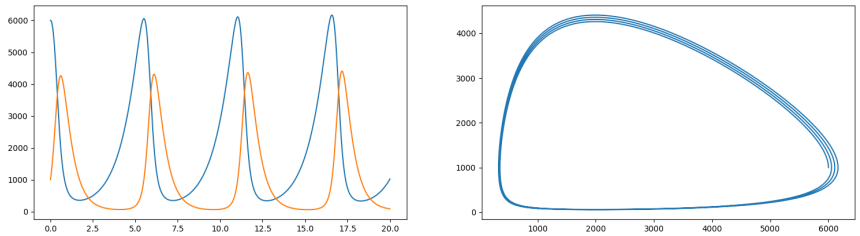
```
>>> np.meshgrid([1, 2, 3], [4, 5, 6, 7])
[array([[1, 2, 3],
        [1, 2, 3],
        [1, 2, 3],
        [1, 2, 3]]),
array([[4, 4, 4],
        [5, 5, 5],
        [6, 6, 6],
        [7, 7, 7]])]
```

4 Solutions

Solution de l'exercice XXII.1 -

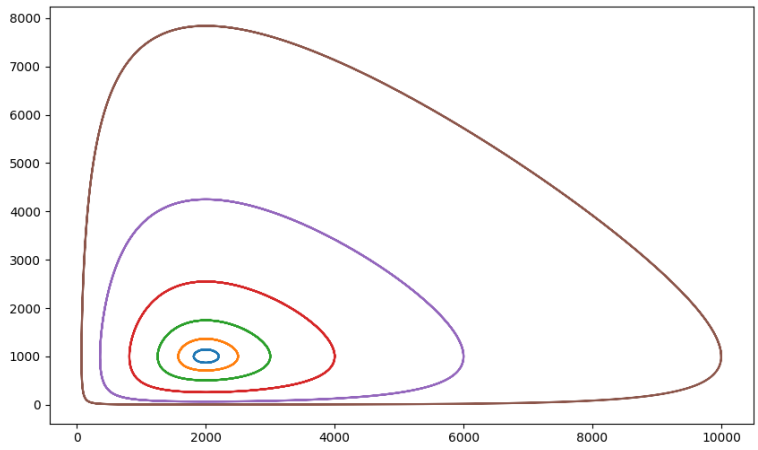
```
def phi(u, t):
    p, q = u
    vp = alpha*p - gamma*p*q
    vq = beta*q + delta*p*q
    return np.array([vp, vq])

T = np.linspace(0, 20, 10000)
p0 = 6000
q0 = 1000
u0 = np.array([p0, q0])
U = Euler(phi, u0, T)
P = [u[0] for u in U]
Q = [u[1] for u in U]
plt.subplot(1, 2, 1)
plt.plot(T, P)
plt.plot(T, Q)
plt.subplot(1, 2, 2)
plt.plot(P, Q)
plt.show()
```



Solution de l'exercice XXII.2 -

```
for p0 in [2200, 2500, 3000, 4000, 6000, 10000]:
    u0 = np.array([p0, q0])
    U = odeint(phi, u0, T)
    P = [u[0] for u in U]
    Q = [u[1] for u in U]
    plt.plot(P, Q)
plt.show()
```

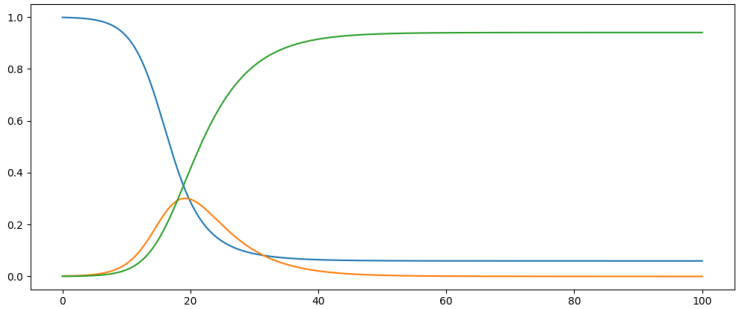


Solution de l'exercice XXII.3 -

```
R0 = 3
gamma = 0.2
beta = R0*gamma

def phi(u, t):
    s, i, r = u
    return np.array([-beta*s*i, beta*s*i - gamma*i, gamma*i])

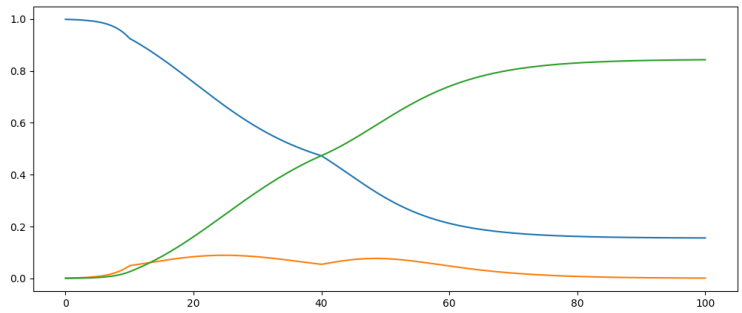
i0 = 1e-3
CI = np.array([1-i0, i0, 0])
t_max = 100
T = np.linspace(0, t_max, 10000)
U = odeint(phi, CI, T)
plt.plot(T,U)
plt.show()
```



Solution de l'exercice XXII.4 -

```
def phi(u, t):
    if 10 < t < 40:
        bet = beta/2
    else:
        bet = beta
    s, i, r = u
    return np.array([-bet*s*i, bet*s*i - gamma*i, gamma*i])

i0 = 0.01
CI = np.array([1 - i0, i0, 0])
t_max = 100
T = np.linspace(0, t_max, 10000)
U = odeint(phi, CI, T)
plt.plot(T,U)
plt.show()
```



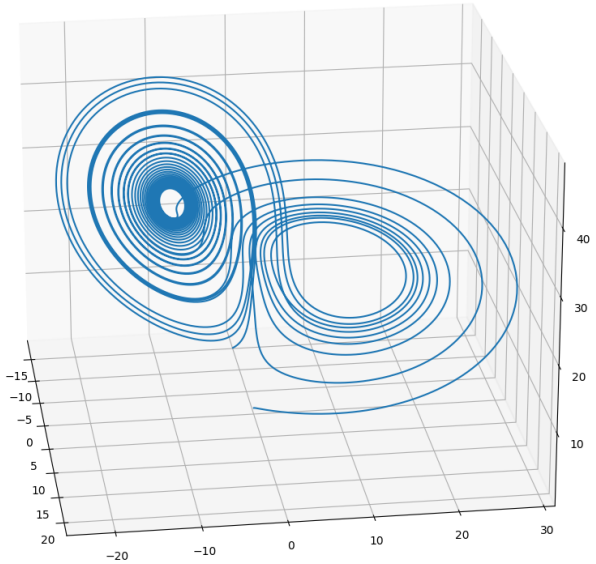
Solution de l'exercice XXII.5 -

```
def lorenz(u, t):
    sigma = 10
    rho = 28
    beta = 8/3
    x, y, z = u
    vx = sigma*(y - x)
    vy = rho*x - y - x*z
    vz = x*y - beta*z
    return np.array([vx, vy, vz])

T = np.linspace(0, 30, 10000)
y0 = np.array([0.1, 0.0, 0.1])
U = odeint(lorenz, y0, T)
X = [u[0] for u in U]
Y = [u[1] for u in U]
Z = [u[2] for u in U]

fig = plt.figure()
mon_dessin = Axes3D(fig, auto_add_to_figure = False)
fig.add_axes(mon_dessin)

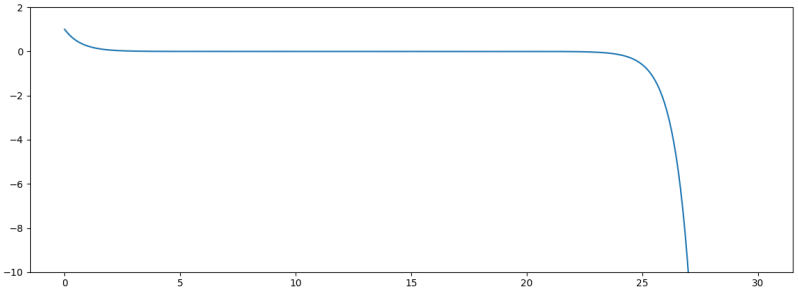
mon_dessin.plot(X, Y, Z)
plt.show()
```



Solution de l'exercice XXII.6 - La solution générale est $A.e^{\sqrt{2}t} + B.e^{-\sqrt{2}t}$. Les conditions initiales imposent $y(t) = e^{-\sqrt{2}t}$ mais les erreurs d'arrondi et d'imprécision font que la solution calculée comporte un terme en $e^{\sqrt{2}t}$ avec un coefficient très petit mais dont le produit par e^{2t} finit par ne plus être négligeable.

```
def phi(u, t):
    y, v = u
    a = 2*y
    return np.array([v, a])

T = np.linspace(0, 30, 10000)
u0 = np.array([1, -2**0.5])
U = odeint(phi, u0, T)
Y = [u[0] for u in U]
plt.plot(T, Y)
plt.ylim([-10, 2])
plt.show()
```



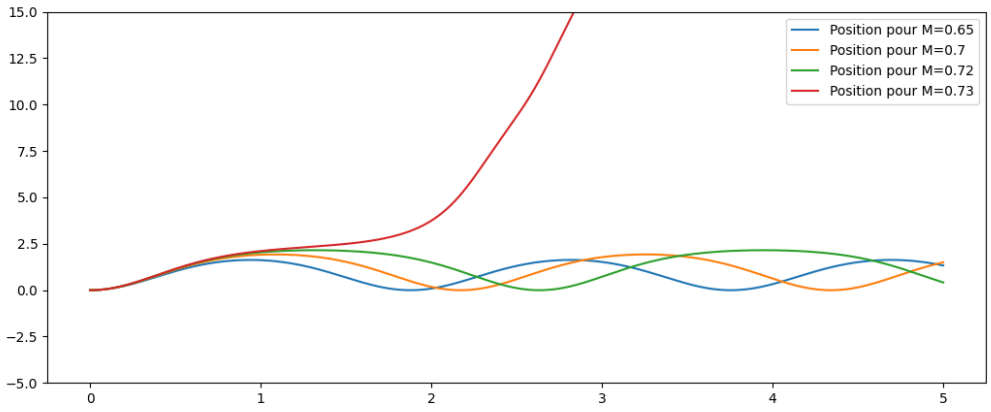
Solution de l'exercice XXII.7 -

```
def phi(u, t):
    y, v = u
    a = (M*g*R - m*g*l*np.sin(y)) / (m*l**2 + M*R**2)
    return np.array([v, a])

N = 10000
tmin = 0
tmax = 5

CI = np.array([0, 0])
T = np.linspace(tmin, tmax, N)
liste_M = [0.65, 0.70, 0.72, 0.73]
for M in liste_M:
    U = odeint(phi, CI, T)
    Y = [u[0] for u in U]
    plt.plot(T, Y, label = "Position pour M={}".format(M))

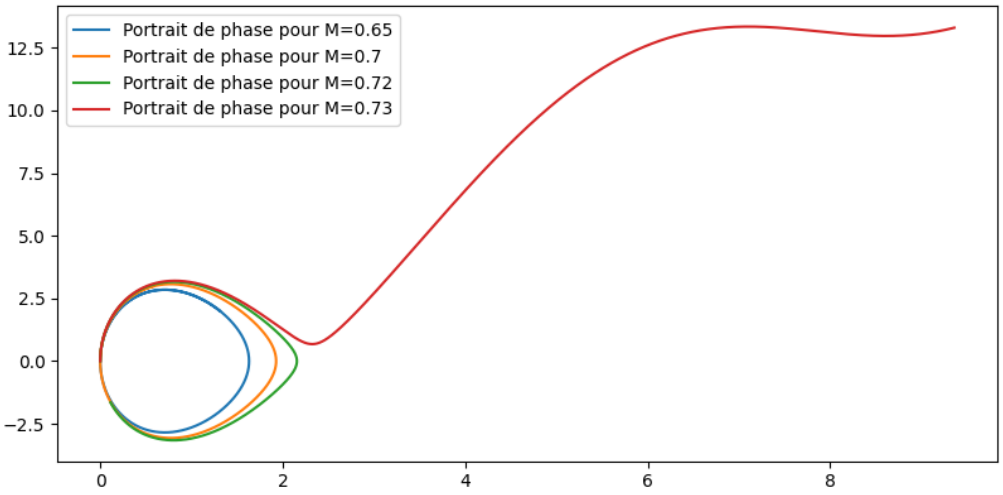
plt.legend()
plt.show()
```



Solution de l'exercice XXII.8 -

```
tmax = 2.5
CI = np.array([0, 0])
T = np.linspace(tmin, tmax, N)
liste_M = [0.65, 0.72, 0.74]
liste_M = [0.65, 0.70, 0.72, 0.73]
for M in liste_M:
    U = odeint(phi, CI, T)
    Y = [u[0] for u in U]
    V = [u[1] for u in U]
    plt.plot(Y, V, label = "Position pour M={}".format(M))

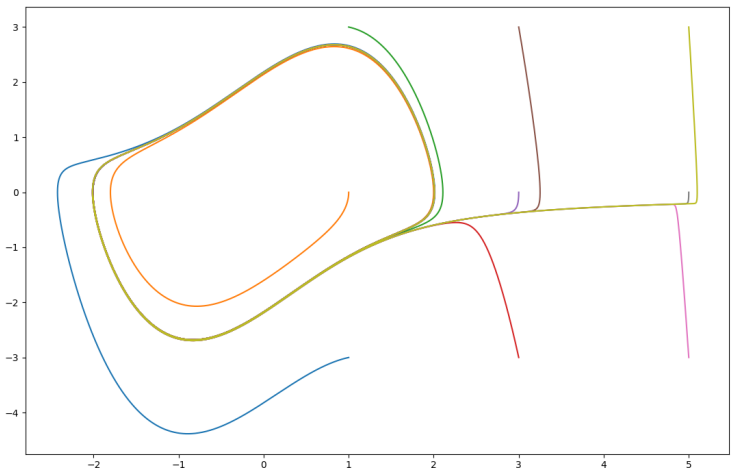
plt.legend()
plt.show()
```



Solution de l'exercice XXII.9 -

```
def phi(u, t):
    y, v = u
    a = epsilon*omega0*(1-y**2)*v -omega0**2*y
    return np.array([v, a])

T = np.linspace(0, 20, 10000)
for y0 in [1, 3, 5]:
    for v0 in [-3, 0, 3]:
        u0 = np.array([y0, v0])
        U = odeint(phi, np.array(u0), T)
        Y = [u[0] for u in U]
        V = [u[1] for u in U]
        plt.plot(Y, V)
plt.show()
```



Solution de l'exercice XXII.10 -

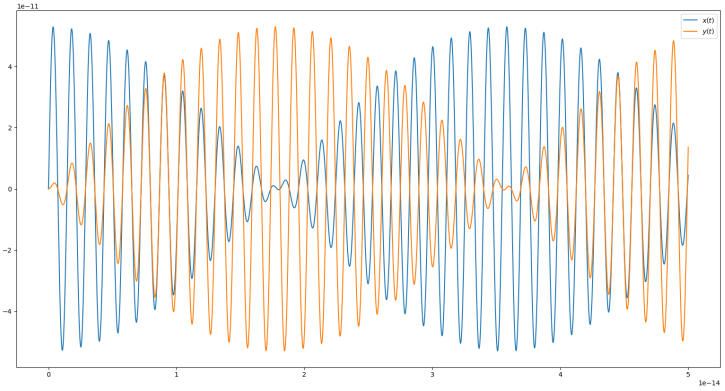
```
def zeeman(u, t):
    x, y, vx, vy = u
    ax = -2*w*vy-w0**2*x
    ay = 2*w*vx-w0**2*y
    return np.array([vx, vy, ax, ay])

t_max = 5e-14
T = np.linspace(0, t_max, 3000)

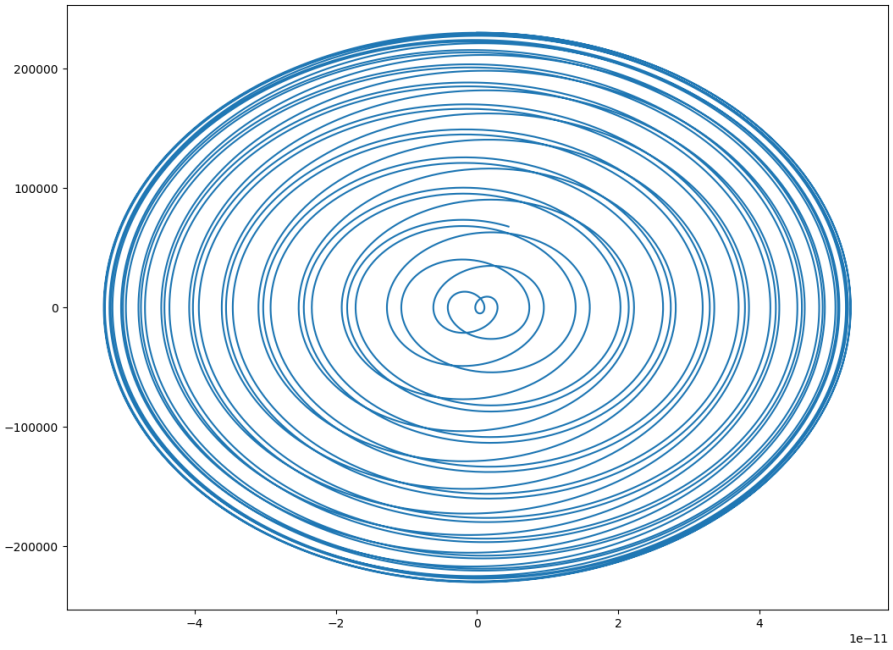
r0x = 0
r0y = 0
v0x = a*w0
v0y = 0
CI = np.array([r0x, r0y, v0x, v0y])


U = odeint(zeeman, CI, T)
X = [u[0] for u in U]
Y = [u[1] for u in U]
VX = [u[2] for u in U]
VY = [u[3] for u in U]

plt.plot(T, X, label = "$x(t)$")
plt.plot(T, Y, label = "$y(t)$")
plt.legend()
plt.show()
```




```
plt.plot(X, VX)
plt.show()
```



Solution de l'exercice XXII.11 -

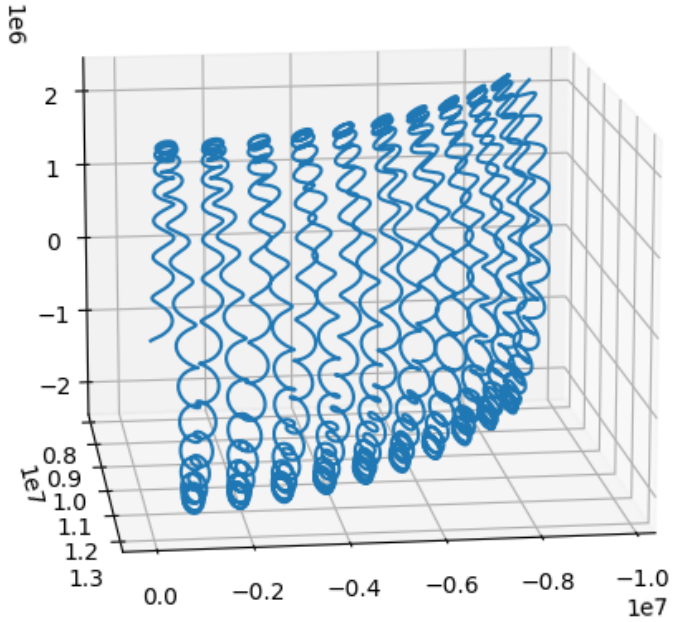
```
def B(M):
    r = np.dot(M, M)**0.5
    u = (1/r)*M
    return mu0*(3*np.dot(mu,u)*u -mu)/(r**3)
```

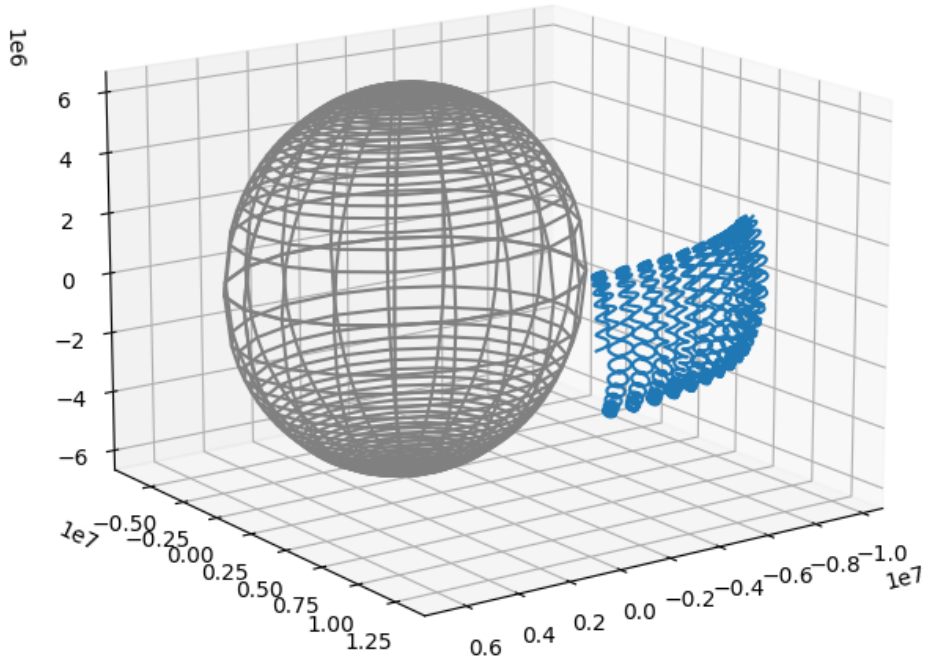
Solution de l'exercice XXII.12 -

```
def phi(u, t):
    M = u[0:3]
    V = u[3:6]
    dM = np.zeros(6)
    dM[0:3] = V
    dM[3:6] = q0*np.cross(V, B(M))/m0
    return dM
```

Solution de l'exercice XXII.13 -

```
T = np.linspace(0, 5, 10000)
u0 = np.array([0, 2*RT, 0, 0, 0.25*c, 0.1*c])
U= odeint(phi, u0, T)
X = U[ : , 0]
Y = U[ : , 1]
Z = U[ : , 2]
fig=plt.figure()
ax=Axes3D(fig, auto_add_to_figure = False)
fig.add_axes(ax)
ax.plot(X, Y, Z)
plt.show()
```


1e6
2
1
0
−1
−2
0.8
0.9
1.0
1.1
1.2
1.3
1e7
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
1e7

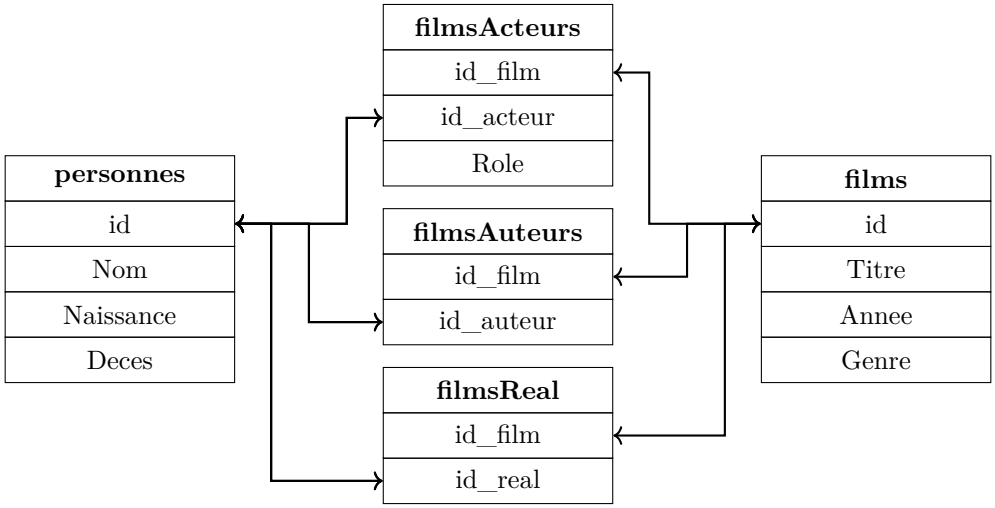

1e6
6
4
2
0
−2
−4
−6
1e7
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
1e7

TP XXIII

SQL : JOINTURES

1 Introduction

Nous allons utiliser une base de données concernant les films.
Les données sont issues d'un projet collaboratif, IMDb (internet movie database).
Pour des raisons de taille la base est restreinte aux films sortis avant 1950.
Elle est composée de 5 tables.



- **films**, qui donne des renseignements sur les films.
  `id` : un identifiant unique de chaque film
  `Titre` : le titre du film
  `Annee` : l'année de sortie du film
  `Genre` : le genre du film
- **personnes**, qui donne des renseignement sur ceux qui ont travaillé sur le film
  `id` : un identifiant unique de chaque personne
  `Nom` : le prénom et le nom
  `Naissance` : l'année de naissance
  `Deces` : l'année de décès (-1 si la personne est en vie)
- **filmsActeurs**, qui indique les acteurs principaux d'un film
  `id_film` : l'identifiant du film
  `id_acteur` : l'identifiant de l'acteur
  `Role` : le rôle joué par l'acteur
- **filmsAuteurs**, qui indique les auteurs d'un film
  `id_film` : l'identifiant du film
  `id_acteur` : l'identifiant de l'auteur
- **filmsReal**, qui indique le(s) réalisateur(s) d'un film
  `id_film` : l'identifiant du film
  `id_real` : l'identifiant du réalisateur

1. Les tables **films** et **personnes** sont des tables d'*entités*, elles décrivent des personnes ou des objets. On peut ajouter des caractéristiques au fur et à mesure que l'on en a connaissance.
2. Les tables **filmsActeurs**, **filmsAuteurs** et **filmsReal** sont des tables de *relations*, elles associent des entités.
3. On a regroupé les descriptions des auteurs, acteurs et réalisateurs dans une même table ; les caractéristiques sont indépendantes des actions qu'ils peuvent réaliser. Une même personne peut d'ailleurs intervenir de différentes façons.
4. Les égalités d'identifiants symbolisées par les flèches ci-dessus sont structurelles ; elles **devront** être utilisées pour faire des jointures et non dans des clauses `WHERE`.
5. Même si la base de données a été réduite elle reste de grande taille : les réponses aux requêtes peuvent être longues à obtenir (de l'ordre de plusieurs secondes).
6. Il est **indispensable** de copier la base sur l'ordinateur, en local, et d'utiliser cette copie. Dans le cas contraire les requêtes auront un temps de réponse prohibitif.

2 Rappels

2.1 Requêtes simples

Exercice XXIII.1
Quels sont les films marqués par le genre `"Film-Noir"` *? (Il y en a 28.)*

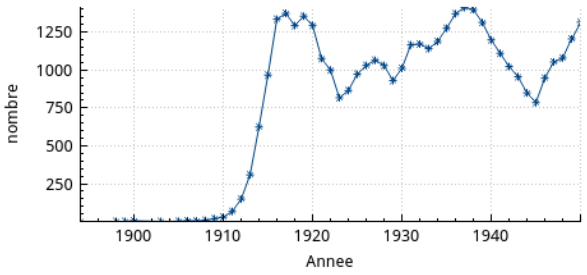
Exercice XXIII.2
Quelles sont les personnes répertoriées qui sont nées avant 1500 ? (Il y en a 10.)
On peut remarquer une erreur de la table si on demande l'année de décès.

Exercice XXIII.3
Quels sont les films dont le titre contient Sherlock Holmes ? (Il y en a 18.)
`like` *pour la ressemblance, % pour une chaîne quelconque.*

Exercice XXIII.4
Quels sont les films dont le titre contient le mot "Napoleon" ? Il y en a 16.

2.2 Agrégations

Exercice XXIII.5
Combien y-a-t-il de films par année ?
`sqlitebrowser` *permet de visualiser le résultat graphiquement : dans le menu* `Vue`*, activer* `Graphique`*.*
Dans l'onglet `graphique` *on peut sélectionner* **x** *et* **y** *avec les attributs utiles.*



Exercice XXIII.6
Combien y-a-t-il de films par genre ?

Exercice XXIII.7
Quels sont les titres qui sont partagés par le plus de films ?
2 titres sont partagés par 9 films.

Exercice XXIII.8
Écrire une requête qui donne uniquement les titres qui réalisent le maximum.

3 Jointures de deux tables

3.1 Jointures simples

Exercice XXIII.9
Quels sont les rôles joués dans le Magicien d'Oz ("The Wizard of Oz") ? Il y en a 4.

Exercice XXIII.10
Quels sont les films dans lesquels un personnage a un nom qui contient "Napoleon" (dans la base les noms n'ont pas d'accent) ? Il y en a 34.

Exercice XXIII.11
Quels sont les films dans lesquels un personnage a un nom qui contient "Napoleon" (dans la base les noms n'ont pas d'accent) ? Il y en a 33.

Exercice XXIII.12
Quels sont les acteurs qui ont joué Sherlock Holmes ?
Il y en a 17 distincts.

Exercice XXIII.13
Quels sont les acteurs ayant joué au moins un rôle contenant le mot "Sheriff" ?
Il y en a 254 distincts.

Exercice XXIII.14
Quels sont les films dont le titre contient le mot "Napoleon" mais pour lesquels n'est pas indiqué de personnage dont le nom contient "Napoleon" ou qui n'ont pas de personnage indiqué ? Il y en a 10.

3.2 Jointures avec agrégations

Exercice XXIII.15
Un acteur a joué le rôle de Sherlock Holmes plus de 10 fois ; qui est-il ?

Exercice XXIII.16
Déterminer, pour chaque année, le nombre d'acteurs ayant joué dans au moins un film. Pour compter le nombre d'acteurs et non le nombre de rôles joués on pourra utiliser `COUNT(DISTINCT filmsActeurs.id_acteur)`

Exercice XXIII.17
Parfois des acteurs jouent des rôles avec leur vrai nom.
Quels acteurs on ainsi joué dans plus de 20 films en portant leur nom ? Il y en a 3.

Exercice XXIII.18
Quels sont les réalisateurs crédités de plus de 100 films dans la base ? Il y en a 17.

Exercice XXIII.19
Quel est le nombre maximal de films réalisés par un réalisateur en une année ?
On ne demande pas ici le nom du réalisateur. On supposera qu'il n'y a pas d'ex-æquo.

4 Jointures de 3 tables ou plus

Dans la structure entités-relations on voit que les entités sont séparées par une relation (au moins). Si on veut lier les noms des acteurs aux titres des films on doit donc écrire une double jointure. On peut écrire une double jointure sous une des deux formes suivantes

```
table1 AS t1 JOIN table2 AS t2 ON t1.attribut1 = t2.attribut2
             JOIN table3 AS t3 ON t1.attribut3 = t3.attribut4

table1 AS t1 JOIN table2 AS t2 JOIN table3 AS t3
             ON     t1.attribut1 = t2.attribut2
                AND t1.attribut3 = t3.attribut4
```

4.1 Requêtes sans agrégation

Exercice XXIII.20
Quels sont les films réalisés par Abel Gance ? Il y en a 21.

Exercice XXIII.21
Quels sont les films dans lesquels a joué Ronald Reagan ? Il y en a 10.

Exercice XXIII.22
Quels sont les films dans lesquels Charles K. French a joué un rôle indiqué comme Sheriff (The Sheriff, Sheriff Nelson, ...) ? Il y en a 8

Exercice XXIII.23
Quels sont les films dans lesquels le réalisateur joue ? Il y en a 1056.

Exercice XXIII.24
Dans quels films ont joué ensemble Fred Astaire et Ginger Rogers ? Il y en a 8.

4.2 Requêtes avec agrégation

Exercice XXIII.25
À la question **XXIII.18** *on peut voir que Sam Newfield est crédité de 191 films.*
Déterminer le nombre de films qu'il a réalisé par année.

Exercice XXIII.26
Déterminer le nom du réalisateur qui a réalisé le nombre maximal de films en un an (question **XXIII.19***).*

Exercice XXIII.27
Calculer la moyenne annuelle du nombre de films réalisés par Sam Newfield entre 1936 et 1946 (limites comprises) ?

Exercice XXIII.28
Quels sont les 3 "couples" réalisateur/acteur ayant fait le plus grand nombre de films en communs ?

Exercice XXIII.29 **
Quels sont les "couples" réalisateur/acteur ayant fait les 4 plus grands nombres de films en communs ? On veut tous les ex-æquo (ils sont 4) pour la quatrième position.

Exercice XXIII.30 **
Déterminer l'acteur (ou les acteurs en cas d'ex-æquos) qui a joué le plus de films chaque année.

5 Complément : championnat de France

Nous allons travailler avec une base de données qui recense les renseignements sur les matchs de division 1 d'une année. La base ne contient qu'une table, nommée `statFoot`.

En voici quelques lignes, la saison est 2020-21

jour	equD	equE	butsD	butsE
21/08/2020	Bordeaux	Nantes	0	0
22/08/2020	Dijon	Angers	0	1
22/08/2020	Lille	Rennes	1	1
23/08/2020	Monaco	Reims	2	2
23/08/2020	Lorient	Strasbourg	3	1
23/08/2020	Nimes	Brest	4	0
...	...	...	...	...

Il y a 20 équipes ; chaque équipe reçoit toutes les autres une fois et une seule.

- `jour` est la date du match, sous forme d'une chaîne de caractères.
- Les autres attributs vont par paire : le suffixe `D` indique que l'attribut concerne l'équipe qui joue à domicile, le suffixe `E` indique que l'attribut concerne l'équipe qui joue à l'extérieur.
- `equD` ou `equE` indique le nom de l'équipe, sous forme d'une chaîne de caractères.
- `butsD` ou `butsE` indique le nombre de buts marqués pendant le match.

Exercice XXIII.31
Écrire une requête qui permet de connaître le nombre de matchs gagnés par chaque équipe et le total des points de l'année.
Un match nul vaut 1 point, un match gagné vaut 3 points.
N.B. Bien que la base ne contienne qu'une table, la requête fait intervenir plusieurs jointures.

6 Solutions

Solution de l'exercice XXIII.1 -

```
select Titre
from films
where Genre = "Film-Noir"
```

Solution de l'exercice XXIII.2 -

```
select Nom, Naissance, Deces
from personnes
where Naissance < 1500
```

Ovide est marqué comme encore vivant.

Solution de l'exercice XXIII.3 -

```
select Titre
from films
where Titre like "%Sherlock Holmes%"
```

Solution de l'exercice XXIII.4 -

```
select titre
from films
where titre like "%Napoleon%"
```

Solution de l'exercice XXIII.5 -

```
select annee, count() as nombre
from films
group by annee
```

Solution de l'exercice XXIII.6 -

```
select genre, count() as nombre
from films
group by genre
```

Solution de l'exercice XXIII.7 -

```
Select Titre, Count() as nb
From Films
Group by Titre
Order by nb desc
```

Solution de l'exercice XXIII.8 -

```
Select Titre, nb
From (Select Titre, Count() as nb From Films Group by Titre)
where nb = (Select Max(nb)
            From (Select Titre, Count() as nb
                  From Films
                  Group by Titre))
```

Solution de l'exercice XXIII.9 -

```
select fa.Role
from films as f join filmsActeurs as fa on f.id = fa.id_film
where f.Titre = "The Wizard of Oz"
```

Solution de l'exercice XXIII.10 -

```
select distinct f.titre
from films as f join filmsActeurs as fa on f.id = fa.id_film
where fa.role like "%Napoleon%"
```

Solution de l'exercice XXIII.11 -

```
select distinct f.titre
from films as f join filmsActeurs as fa on f.id = fa.id_film
where fa.role like "%Napoleon%"
```

Solution de l'exercice XXIII.12 -

```
select distinct p.nom
from filmsActeurs as fa join personnes as p on p.id = fa.
   id_acteur
where fa.Role = "Sherlock Holmes"
```

Solution de l'exercice XXIII.13 -

```
select distinct p.nom, fa.role
from filmsActeurs as fa join personnes as p on p.id = fa.
   id_acteur
where fa.Role like "%heriff%"
```

Solution de l'exercice XXIII.14 -

```
select titre
from films
where titre like "%Napoleon%"
EXCEPT
select distinct f.titre
from films as f join filmsActeurs as fa on f.id = fa.id_film
where fa.role like "%Napoleon%"
```

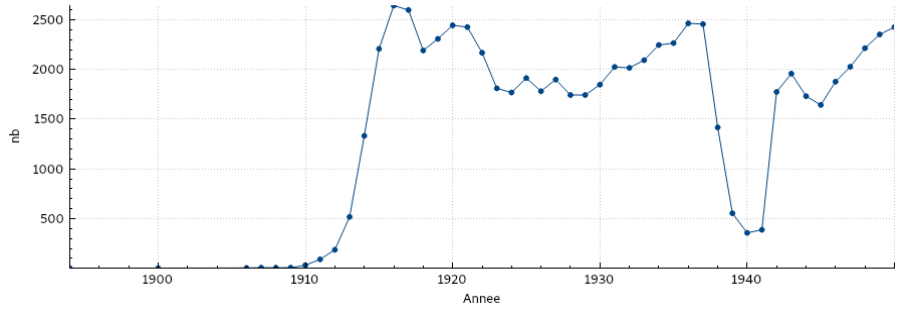
Solution de l'exercice XXIII.15 -

```
select distinct p.nom, count() as nombre
from filmsActeurs as fa join personnes as p on p.id = fa.
    id_acteur
where fa.Role = "Sherlock Holmes"
group by p.nom
having nombre >= 10
```

C'est Basil Rathbone.

Solution de l'exercice XXIII.16 -

```
select f.Annee, count(distinct fa.id_acteur) as nb
from films as f join filmsActeurs as fa on fa.id_film = f.id
group by f.annee
```



Solution de l'exercice XXIII.17 -

```
select distinct p.nom, count() as nombre
from filmsActeurs as fa join personnes as p on p.id = fa.
    id_acteur
where fa.Role = p.nom
group by p.nom
having nombre > 20
```

Solution de l'exercice XXIII.18 -

```
select p.nom, count() as nombre
from personnes as p join filmsReal as fr on p.id = fr.id_real
group by p.nom
having nombre > 100
```

Solution de l'exercice XXIII.19 -

```
select f.annee, count() as nombre
from films as f join filmsReal as fr on f.id = fr.id_film
group by f.annee, fr.id_real
order by nombre DESC
limit 1
```

Solution de l'exercice XXIII.20 -

```
select Titre, Annee
from personnes as p join filmsReal as fr on p.id = fr.id_real
                    join films as f on f.id = fr.id_film
where p.nom = "Abel Gance"
```

Solution de l'exercice XXIII.21 -

```
select f.titre, f.annee
from films as f join filmsActeurs as fa on f.id = fa.id_film
                join personnes as p on p.id = fa.id_acteur
where p.nom = "Ronald Reagan"
```

Si on filtre avec `p.nom like "%Reagan"` on trouve deux films en plus dans lesquels joue Nancy Reagan.

Solution de l'exercice XXIII.22 -

```
select f.titre, fa.role, f.genre
from films as f join filmsActeurs as fa on f.id = fa.id_film
                join personnes as p on p.id = fa.id_acteur
where fa.role like "%Sheriff%" and p.nom = "Charles K. French"
```

Solution de l'exercice XXIII.23 -

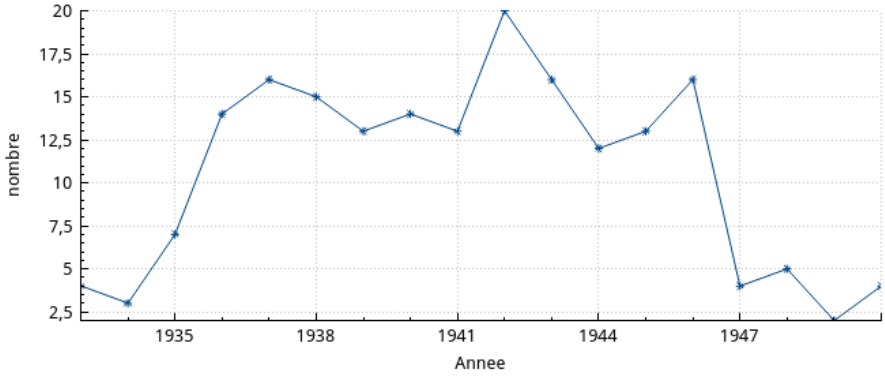
```
select pr.nom, f.titre
from films as f join filmsActeurs as fa on f.id = fa.id_film
                join filmsReal as fr on f.id = fr.id_film
                join personnes as pr on pr.id = fr.id_real
                join personnes as pa on pa.id = fa.id_acteur
where pr.id = pa.id
```

Solution de l'exercice XXIII.24 -

```
select f.titre, f.annee
from films as f join filmsActeurs as fa on f.id = fa.id_film
                join  personnes as p on p.id = fa.id_acteur
where p.nom = "Fred Astaire"
INTERSECT
select f.titre, f.annee
from films as f join filmsActeurs as fa on f.id = fa.id_film
                join  personnes as p on p.id = fa.id_acteur
where p.nom = "Ginger Rogers"
```

Solution de l'exercice XXIII.25 -

```
select Annee, count() as nombre
from personnes as p join filmsReal as fr on p.id = fr.id_real
                    join films as f on f.id = fr.id_film
where p.nom = "Sam Newfield"
group by Annee
order by Annee
```



Solution de l'exercice XXIII.26 -

```
select f.annee, p.nom, count() as nombre
from films as f join filmsReal as fr on f.id = fr.id_film
                join personnes as p on p.id = fr.id_real
group by f.annee, fr.id_real
order by nombre DESC
limit 1
```

Solution de l'exercice XXIII.27 -

```
select avg(nombre)
from (select Annee, count() as nombre
      from films as f join filmsReal as fr on f.id = fr.
         id_film
                      join personnes as p on p.id = fr.id_real
      where p.nom = "Sam Newfield"
      group by Annee)
where Annee >= 1936 and Annee <= 1946
```

Solution de l'exercice XXIII.28 -

```
select pr.nom as Réal, pa.nom as Act, count(*) as "Nb films"
from filmsReal as fr join personnes as pr on fr.id_real=pr.id
            join filmsActeurs as fa on fr.id_film=fa.id_film
            join personnes as pa on fa.id_acteur=pa.id
group by pr.id ,pa.id
order by 3 desc
limit 3
```

Solution de l'exercice XXIII.29 -

```
select pr.nom as Réal, pa.nom as Act, count(*) as "Nb films"
from filmsReal as fr join personnes as pr on fr.id_real=pr.id
            join filmsActeurs as fa on fr.id_film=fa.id_film
            join personnes as pa on fa.id_acteur=pa.id
group by pr.id ,pa.id
having count() >= (select distinct count()
                   from filmsActeurs as fa
                           join filmsReal as fr
                           on fr.id_film=fa.id_film
                   group by fr.id_real,fa.id_acteur

                   order by 1 desc  limit 1  offset 3)
order by 3 desc
```

Solution de l'exercice XXIII.30 -

```
select m.an, n.nom, m.nmax
from (select an, max(nb) as nmax
      from (select f.annee as an, p.nom as nom, count() as nb
            from films as f join filmsActeurs as fr
                                on f.id = fr.id_film
                            join personnes as p
                                on p.id = fr.id_acteur
                            group by f.annee, fr.id_acteur)
      group by an) as m
      join (select f.annee as an, p.nom as nom, count() as nb
            from films as f join filmsActeurs as fr
                                on f.id = fr.id_film
                            join personnes as p
                                on p.id = fr.id_acteur
            group by f.annee, fr.id_acteur) as n
      on m.an = n.an
where n.nb = m.nmax
```

Solution de l'exercice XXIII.31 -

```
select d.club, (d.victoiresD + e.victoiresE) as victoires,
      (3*(d.victoiresD + e.victoiresE) + n.nuls) as points
from (select equD as club, count() as victoiresD
      from stat where butsD > butsE group by equD) as d
     join
     (select equE as club, count() as victoiresE
      from stat where butsD < butsE group by equE) as e
     join
     (select equD as club, count() as nuls
      from stat where butsD = butsE group by equD) as n
     on e.club = d.club and e.club = n.club
order by points desc
```