# Résumé du programme de 1ère année.

MP Lycée Camille Jullian

mars 2022

Le but de ce document est de résumer très succintement les éléments du programme de première année qu'on n'a pas trop revus ensemble histoire que vous ayez facilement sous la main les points les plus importants à maîtriser. Attention tout de même, ce n'est absolument pas détaillé, je vous laisse relire vos cours de l'an dernier pour quelque chose de plus complet. J'ai par ailleurs volontairement mis de côté les bases du Python (boucles, manipulations de listes) qu'on a suffisamment retravaillées ensemble depuis septembre.

## 1 Représentation des nombres.

Toutes les variables Python sont bien entendus à la base des entiers binaires. Plus précisément, les conventions habituellement utilisées pour stocker en mémoire des valeurs numériques sont les suivantes :

- les entiers naturels sont codés en binaire de façon naturelle, avec une borne supérieure dépendant du nombre de bits utilisés pour le stockage. Ainsi, sur 16 bits, on pourra stocker les entiers compris entre 0 et $2^{16} - 1 = 65\ 535$.
- les entiers relatifs sont codés en complément à 2, c'est-à-dire qu'on code les entiers entre $-2^{n-1}$ et $2^{n-1} - 1$ (où $n$ est le nombre de bits) modulo $2^n$. Tous les nombres ayant un premier bit égal à 1 correspondent donc aux entiers négatifs (principe du bit de signe).
- les flottants (nombres réels) sont décomposés en trois parties : un bit de signe (1 pour un réel négatif), un certain nombre de bits pour coder un exposant entier reatif (en complément à 2, donc), et le reste (la mantisse) pour coder un entier naturel représentant les chiffres significatifs (binaires) du nombre codé. En notant $e$ l'exposant et $m$ la mantisse, on code ainsi le réel $\pm m \times 2^e$.

## 2 Bases de données.

Une base de données relationnelle est constituée d'une liste de tables (ou relations) qu'on peut voir comme des tableaux de données, chaque ligne représentant un attribut de la relation (caractéristique qui doit donner lieu dans la table à la création d'une donnée unique). Parmi ces attributs, l'un joue le rôle de clé primaire : c'est un attribut qui prend des valeurs différentes pour chaque instance de la table, et permet donc d'identifier de façon unique chaque ligne à l'intérieur du tableau (parfois, c'est un couple d'attributs, voire pire, qui joue ce rôle). Les différentes tables sont reliées entre elles par le principe des clés secondaires : une table peut avoir comme attribut la clé primaire d'une autre table (mais qui ne joue pas le rôle de clé primaire dans cette table). Par exemple, dans une base de données consacrée au cinéma, on aura une table **Réalisateurs** contenant (entre autres) comme attributs un numéro d'identification (clé primaire), le nom, le prénom etc du réalisateur. Dans une autre table **Films**, on retrouvera comme attribut (en plus de beaucoup d'autres) le numéro d'identification du réalisateur, qui jouera ici le rôle de clé secondaire.

Une petite liste de commandes SQL à connaître pour effectuer des recherches ou des mises à jour dans une base de données relationnelle :

- CREATE TABLE nom (attribut1 type1, attribut2 type2, ...)
  sert à créer une table en donnant la liste de ses attributs ainsi que le type de données correspondant.
- ALTER TABLE nom ADD attribut type
  sert à ajouter un attribut à une table déjà existante (en remplaçant le ADD par DROP on peut aussi supprimer une colonne).
- INSERT INTO table VALUES(att1,att2,...)
  sert à insérer de nouvelles données dans une table (les valeurs indiquées entre parenthèses doivent être de types conformes à ceux déclarés lors de la création de la table).
- UPDATE table SET condition WHERE condition2
  sert à mettre à jour une table, en modifiant toutes les lignes vérifiant la condition2 pour leur imposer la condition indiquée après la commande SET.
- SELECT attribut FROM table WHERE condition
  sert à effectuer dans une table la recherche de toutes les lignes vérifiant une certaine condition, en n'affichant que la valeur des attributs indiqués après la commande SELECT (s'il y en a plusieurs, on les sépare par des virgules).
- on peut ajouter à la suite d'un SELECT des options du type ORDER BY attribut pour ordonner les résultats suivant la valeur de l'attribut (ordre croissant par défaut, décroissant en ajoutant un DESC derrière l'attribut), ou GROUP BY pour les regrouper suivant la valeur d'un certain attribut.
- SELECT attribut FROM table1 JOIN table2 ON att1=att2 WHERE condition
  permet d'effectuer une recherche impliquant deux tables dont on fera la jointure en imposant l'égalité des attributs att1 et att2. On peut bien sûr étendre le principe à plus de deux table. Si les attributs att1 et att2 portent le même nom, on se contentera d'utiliser un NATURAL JOIN sans le ON. Si le nom de l'attribut est ambigu (il existe un attribut portant ce nom dans les deux tables, on le fait précéder du nom de la table sous la forme table.att).
- SELECT COUNT(*) FROM table WHERE condtion
  permet de compter le nombre de lignes de la table vérifiant une certaine condition. On peut également utiliser d'autres fonction, comme SUM (fera la somme des résultats, à appliquer à un attribut numérique), MAX, MIN ou AVG (maximum, minimum, moyenne, toujours pour des attributs numériques).

## 3 Résolution d'équations.

Pour résoudre de façon approchée une équation du type $f(x) = 0$ (où $f$ est une fonction continue dont on sait qu'elle s'annule sur un certain intervalle), on dispose des outils suivants :

- méthode exhaustive : on calcule des valeurs de la fonction en augmentant à chaque fois la variable d'un certain pas correspondant à la précision souhaitée, jusqu'à observer un changement de signe (méthode très peu efficace).
- dichotomie : on découpe l'intervalle en deux à chaque étape, en conservant la moitié où la fonction change de signe (à partir d'un intervalle de largeur 1, il faudra 10 étapes pour obtenir une précision de $10^{-3}$, 20 étapes pour $10^{-6}$ etc).
- méthode de Newton : on part d'un point de la courbe, puis à chaque étape, on calcule l'intersection de la tangente en ce point avec l'axe des abscisses, ce qui revient à calculer $x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$. Beaucoup plus rapide, mais nécessite une fonction dérivable (et un choix de $x_0$ pas trop idiot).

# 4 Calcul numérique d'intégrales.

Pour calculer la valeur approchée de $I = \int_a^b f(t)\ dt$, on découpe l'intervalle en morceaux en posant $x_k = a + k\dfrac{b-a}{n} = a + kh$, en ayant posé $h = \dfrac{b-a}{n}$ (largeur de chaque morceau), puis on utilise l'une des formules suivantes :

- méthode des rectangles : $I \simeq h\sum_{k=1}^{n} f(x_k)$ ou $I \simeq h\sum_{k=0}^{n-1} f(x_k)$
- méthode des trapèzes : $I \simeq h\sum_{k=0}^{n-1} \dfrac{f(x_k)+f(x_{k+1})}{2} = \dfrac{f(a)+f(b)}{2} + \sum_{k=1}^{n-1} f(x_k)$
- méthode de Simpson : $I \simeq \dfrac{h}{6}\sum_{k=0}^{n-1} f(x_k) + 4f\left(\dfrac{x_k+x_{k+1}}{2}\right) + f(x_{k+1})$

  $= h\left(\dfrac{f(a)+f(b)}{6} + \dfrac{1}{3}\sum_{k=1}^{n-1} f(x_k) + \dfrac{2}{3}\sum_{k=0}^{n-1} f\left(\dfrac{x_k+x_{k+1}}{2}\right)\right)$

# 5 Résolution approchée d'équations différentielles

On applique la méthode d'Euler, qui consiste à discrétiser l'équation, puis à approcher la courbe sur chaque intervalle par une droite tangente (approchée). En pratique, après avoir mis l'équation sous la forme $y' = F(y,t)$, on calcule les termes de la suite récurrente définie par $y_0 = y(t_0)$ (condition initiale connue), puis $y_{n+1} = y_n + h \times F(y_n, t_n)$, le pas constant $h$ représentant l'écart entre deux valeurs successives du temps où on veut calculer nos valeurs $y_n$ (ainsi, on aura simplement $t_n = t_0 + n \times h$). Un programme basique effectuant la résolution, avec comme paramètres la fonction à deux variables $F$ décrivant l'équation différentielle, la valeur de $t_0$ ainsi que celle de $t_f$ représentant la borne supérieure de l'intervalle sur lequel on souhaite résoudre l'équation, la valeur initiale $y_0$ et le nombre $n$ de points de calcul souhaité :

```
import matplotlib.pyplot as plt
def Euler(F,t0,tf,y0,n) :
    h=(tf-t0)/n
    temps,fonction=[t0],[y0]
    for i in range(n) :
        fonction.append(fonction[-1]+h*F(fonction[-1],temps[-1])
        temps.append(temps[-1]+h)
    plt.plot(temps,fonction)
    return
```

On peut adapter facilement l'algorithme pour résoudre un système de deux équations différentielles du premier ordre (on aura simplement besoin de deux fonctions $F$ et $G$ pour décrire les deux équations du système, et d'une liste supplémentaire pour stocker les valeurs calculées pour la deuxième fonction inconnue), et en déduire une méthode de résolution approchée pour une équation du second ordre. En effet, une équation du type $y'' = F(y', y, t)$, peut toujours se transformer en système : $\left\{ \begin{array}{rcl} y' & = & z \\ z' & = & F(z,y,t) \end{array} \right.$.