I Commandes de manipulation des listes

1.1 Listes

Rappels de quelques commandes utiles, concernant les listes :

```
L = [1, 12, True]  # crée une liste
L = []  # la liste vide
L.append(1)  # ajoute l'élément donné à la fin de L.
L2 = [2, 3]
L + L2  # concaténation.
L[0]  # accès au premier élément
L[0] = 12  # modification du premier élément
```

II Tracé de courbes

2.1 Principe du tracé en python

La fonction de tracé que nous allons utiliser est dans une bibliothèque qu'il faut charger au préalable.

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

Après l'exécution de cette commande (directement dans la console, ou au début d'une feuille de script), la bibliothèque `matplotlib.pyplot` as chargée en mémoire et on peut accéder aux fonctions qu'elle contient par la syntaxe `plt.la_fonction` Le principe général de tracé de courbes est simple. On passe à la fonction de tracé une liste de points (définis par leurs coordonnées) et la fonction se charge d'afficher ces points et de les relier par des lignes droites.

```
plt.plot(X, Y)
```

où X est la liste des abscisses et Y la liste des ordonnées des points à relier (ces deux listes sont donc obligatoirement de même longueur).

2.2 Mise en pratique

Pour créer une liste en calculant les éléments un par un on utilise très souvent la construction :

```
# L est une liste, par exemple vide
for variable in untruc:
    # ici on calcule l'élément à ajouter
    L.append(element)
```

Si le calcule de chaque élément est très simple, on peut utiliser une liste par compréhension

```
L = [i*(i+1) for i in range(10)]
```

a le même effet que

```
L = []
for i in range(10):
    L.append(i*(i + 1))
```

Exercice 1
Nous allons avoir besoin de créer des listes d'abscisses rapidement. Dans le fichier python **tp9.py** une fonction `abscisses(a, b, n)` est fournie. Elle retourne une liste de $n+1$ points équirépartis $[a, \ldots, b]$. On découpe l'intervalle $[a, b]$ en n segments de longueur égales.

```
def abscisses1(a, b, n):
    l = (b - a) / n
    absc = a
    X = []
    for i in range(n + 1):
        X.append(absc)
        absc = absc + l
    return X
```

Créer une fonction `abscisse2` ayant le même effet, mais utilisant une liste par compréhension. Pour ce faire, vous devez savoir exprimer en fonction de `i` (la variable de boucle) la valeur de `X[i]` dans la fonction précédente.

Exercice 2
Tracer la courbe représentative de sin sur l'intervalle $[-\pi, \pi]$. On prendra une liste d'ordonnées de longueur 100.
Indication : le module `numpy` (voir plus loin) est importé dans la feuille de script avec l'alias `np`. La constante π y est définie comme `np.pi`

Pour varier la présentation, on peut passer d'autres arguments à la fonction plot.

```
plt.plot(X, Y, 'o')  # change le style de tracé
plt.plot(X, Y, color=''cyan'')  # pour imposer la couleur.
```

Si on veut ajouter une légende

```
plt.plot(X, Y, label=''Ma super courbe'')
plt.legend()
```

Exercice 3
Créer (dans la feuille de script) une fonction `trace(f, a, b, n)` qui prend une **fonction** f, deux nombres a, b et un entier naturel non nul n comme paramètre et trace la courbe de f sur l'intervalle $[a, b]$ que l'on a découpé en n segments.

2.2.1 Plusieurs tracés

Exercice 4
Utiliser la fonction précédente pour tracer tan sur $[-\pi, \pi]$. Le résultat fait apparaître des droites qui ne doivent normalement pas apparaître dans cette courbe.
Pour y remédier, on va utiliser plusieurs appels à `plt.plot`.

- Définir une variable $epsilon = 1e-3$.
- Créer les liste `X1, X2, X3` des abscisses dans $[-\pi, -\frac{\pi}{2} - epsilon], [-\frac{\pi}{2} + epsilon, \frac{\pi}{2} - epsilon], [\frac{\pi}{2} + epsilon, \pi]$ de manière à équi-répartir les points et en obtenir 100 en tout.
- Créer les listes `Y1, Y2, Y3` des ordonnées correspondantes, puis utiliser

```
plt.plot(X_1, Y_1, X_2, Y_2, X_3, Y_3)
```

III La bibliothèque numpy

Il se trouve que python dispose d'une bibliothèque très pratique pour effectuer toutes les opérations précédentes.

```
import numpy as np
```

Après l'exécution (une seule fois par session de console suffit) de cette commande, le module `numpy` est chargé sous le nom `np`.

3.1 `linspace`

Exercice 5
Comprendre le fonctionnement de la commande `np.linspace`.
Pour obtenir de l'aide sur cette fonction, on pourra utiliser au choix

```
help(np.linspace)
# ou
np.info(np.linspace)  # seulement pour les fonctions numpy
```

3.2 Les fonctions numpy

Exercice 6
Le module `numpy` contient une version des fonctions mathématiques usuelles

```
np.sin([0, np.pi, np.pi/2])
```

Ces fonctions peuvent d'appliquer à des nombres aussi bien qu'à des listes ou des vecteurs numpy (la version numpy des listes, par exemple la valeur de retour de `linspace`)
En 3 lignes, dans la console, tracer la courbe représentative de exp sur $[-1, 1]$ avec 150 points.

Exercice 7 (Bonus)
Créer une fonction `vectorise(f)` où f est une fonction numérique (c'est à dire qui s'applique à un nombre et retourne un nombre) et qui retourne une **fonction** g. g doit être définie pour des listes de valeurs et retourner la liste des valeurs de f correspondante.

IV Fonctions usuelles

Les vecteurs numpy

Les opérations arithmétiques (+, -, *, /, **, %, //) sont redéfinies pour les vecteurs numpy pour s'appliquer terme à terme. Tester dans la consoles les commandes suivantes.

```
v = np.array([3, 5, 7])
2*v
v + 4
```

4.1 Les solutions d'équations différentielles

Créer une fonction `sol_edl2(omega, A, phi, tf)` qui trace la courbe de la fonction $t \mapsto A\cos(\omega t + \varphi)$ sur l'intervalle $[0, tf]$. On utilisera 250 points pour ce tracé.

4.2 Les fonctions polynomiale

On représente la fonction polynomiale $f : x \mapsto a_0 + a_1 x + \cdots + a_n x^n$ par la liste $[a_0, \ldots, a_n]$.

Exercice 8
créer la fonction `valeur_poly(L, x0)` qui prend comme argument une liste `L` représentant un polynôme f et un nombre `x0` et qui retourne le nombre $f(x0)$.

Exercice 9
Créer une fonction `trace_polynome(L, a, b)` qui trace la courbe représentative de la fonction polynomiale représentée par la liste L sur l'intervalle $[a, b]$ (on utilise 250 points pour le tracé). Votre fonction devra faire appel à la fonction de l'exercice précédent.
Cette fonction devra retourner les deux listes construites.

Exercice 10
Si X est un vecteur numpy, alors les opérations suivantes s'effectuent terme à terme

```
X + X  # somme terme à terme
X*X  # produit terme à terme
X + 4  # ajout de 4 à tous les éléments de X
X**2 + 3*X + 1
```

Reprendre l'exercice précédent en utilisant `np.linspace` et seulement des opérations sur les vecteurs numpy.

V Algorithmes autour des liste de valeurs

Exercice 11
Créer une fonction `extrema(X, Y)` qui retourne une liste à deux éléments : (x_m, y_m) et (x_M, y_M) qui sont les coordonnées du minimum (première occurrence) et du maximum (première occurrence) des valeurs de la fonction représentée par X et Y (sa courbe est obtenue par `plt.plot(X, Y)`).

Exercice 12
Créer une fonction `annulation(X, Y)` qui prend une liste d'abscisses et une liste d'ordonnées et retourne la liste des valeurs approchées des éventuels abscisses où la fonction correspondante s'annule.
On considère, comme pour le tracé python, que les fonctions sont affines par morceaux (et donc s'annulent lorsqu'elles changent de signe, d'après le TVI).

Exercice 13
Créer une fonction `monotone` dont on précisera les arguments et qui vérifie si une fonction est monotone (en restant dans le cadre de ce td). La valeur de retour devra être un booléen.

Exercice 14
Créer une fonction `extrema_locaux` qui retourne deux listes (dans cet ordre) : la liste des indices des minima locaux et la liste des indices des maxima locaux, ces listes pouvant être vide (par exemple dans le cas d'une fonction strictement monotone).