# TD PYTHON 01

## Autour des polynômes de Lagrange

❑ **M thode 0.1.— Comment faire des opérations sur les polynômes**

On peut rentrer un polynôme comme une fonction (voir la methode correspondante).
On peut aussi utiliser la fonction `Polynomial` du sous-module `numpy.polynomial` et un certain nombre d'attributs associés. On commence donc par taper :

```
from numpy.polynomial import Polynomial
```

- Pour ***créer un polynôme***, on liste ses coefficients par ordre de degré croissant. Ainsi pour $p = a0 + a1X + a2X^2 + a_3X^3$, on tape : `p = Polynomial([a0,a1,a2,a3])`
  L'attribut `coef` donne accès aux coefficients ordonnés par degré croissant.
  Ainsi, si l'on tape : `p.coef`, on obtient `array([a0.,a1.,a2.,a3.])`
  ***On suppose dans la suite de la méthode que*** $p$ ***a été défini ainsi.***
- Pour ***calculer la valeur*** $p(a)$, on tape simplement : $p(a)$ et pour obtenir $[p(x1), ..., p(xn)]$, on tape : $p([x1, ..., xn])$.
- `p.coef[i]` fournit le ***coefficient devant*** $X^i$.
- `p.degree( )` donne le ***degré*** et `p.roots( )` les ***racines*** de $p$.
- Pour ***dériver le polynôme*** $p$, on tape : `p.deriv( )` qui renvoie un nouveau polynôme. Cet attribut a un argument optionnel $n$ qui indique le nombre de dérivations à effectuer. Si l'on tape par exemple `p.deriv(2).coef`, l'on obtient les coefficients du polynôme dérivée seconde de $p$.
- Pour ***intégrer le polynôme*** $p$, on tape : `p.integ( )` qui renvoie un nouveau polynôme. Cet attribut a un premier argument optionnel donnant le nombre d'intégrations à effectuer et un second argument optionnel qui donne la constante d'intégration (ou la liste de constantes si l'on intègre plusieurs fois) à utiliser.
- Pour faire des ***opérations*** sur des polynômes, on utilise $+$, $-$ et $*$ pour additionner, soustraire et multiplier des polynômes. Par exemple, l'opération `(p**3).coef` permet d'obtenir la liste des coefficients de $p^3$.
- Pour faire la ***division euclidienne*** du polynôme $p$ par le polynôme $q$, la commande `p // q` donne le polynôme quotient et `p % q` donne le reste.

### Exercice 01 : manipulation de polynômes

Soit le polynôme $P = X^3 + 2X - 3$, calculer successivement $P(0)$, $[P(1), P(2), P(3)]$, deg $P$, les racines de $P$, puis les polynômes $P'$, $P''$ et enfin le quotient et le reste de la division euclidienne de $P$ par $B = X^2 + 1$.

### Cours : polynômes de Lagrange

Soient $n+1$ couples $(x_0, y_0), ..., (x_n, y_n)$ dans $\mathbb{R}^2$ tel que pour tout $i \in [\![0, n]\!]$, les réels $x_i$ soient **tous distincts**. Il s'agit de construire un polynôme $P$ de degré au plus $n$ tel que pour tout $i \in [\![0, n]\!]$, on ait l'égalité : $P(x_i) = y_i$.

**Définition :** *On appelle* ***polynômes interpolateurs de Lagrange*** *associés à la famille* $(x_0, x_1, ..., x_n)$ *les* $n+1$ *polynômes définis pour tout* $i \in [\![0, n]\!]$ *par :*

$$L_i = \prod_{j=0, j\neq i}^{n} \frac{X - x_j}{x_i - x_j}.$$

box 0.90 setgray fill 1 setlinewidth

**Proposition 0.1.— Propriétés des polynômes interpolateurs de Lagrange**

- Pour tout $i \in [\![0, n]\!]$, $L_i$ est de degré $n$.
- Pour tout $i \in [\![0, n]\!]$ et pour tout $j \in [\![0, n]\!]$, $L_i(x_j) = \delta_{i,j} = \begin{cases} 1 & \text{si} \quad j = i \\ 0 & \text{si} \quad j \neq i \end{cases}$
- La famille $(L_0, L_1, ..., L_n)$ est une famille libre de $\mathbb{K}_n[X]$ et en est donc une base. On l'appelle **base d'interpolation de Lagrange** associée à $(x_0, x_1, ..., x_n)$.

Comme $(L_0, ...L_n)$ est une base de $\mathbb{K}_n[X]$, tout polynôme de $\mathbb{K}_n[X]$ s'exprime dans cette base et on a en particulier le résultat suivant.

box 0.90 setgray fill 1 setlinewidth

**Proposition 0.2.—** Il existe un seul polynôme de degré au plus $n$ (donc dans $\mathbb{K}_n[X]$) qui vérifie pour tout $i \in [\![0, n]\!]$, $P(x_i) = y_i$. C'est le polynôme $P = \sum_{i=0}^{n} y_i L_i$.

En particulier, $1 = \sum_{i=0}^{n} L_i$ et $X = \sum_{i=0}^{n} x_i L_i$.

**Exercice 02 : Un premier exemple**

Déterminer la base d'interpolation de Lagrange pour la famille $(0, 1, 2)$.
En déduire l'unique polynôme $P$ de degré 2 tel que $P(0) = 1$, $P(1) = -3$ et $P(2) = 2$.

❑ **M thode 0.2.— Comment construire des polynômes de Lagrange**

On désire construire le polynôme de Lagrange $P$ passant par les points $(x_0, y_0), ..., (x_n, y_n)$.

- On peut vouloir une liste de points où passe $P$ (pas nécessairement la base d'interpolation), voir **l'exercice 04**.
- On peut vouloir $P$ sous la forme d'un polynôme, on utilise alors `numpy.polynomial`, voir **l'exercice 03**.
- On peut vouloir $P$ sous la forme d'une fonction (par exemple pour tracer $P$), voir **l'exercice 04.**
- On peut vouloir $P$ directement en utilisant `interpolate.lagrange` du module `scipy`, voir **l'exercice 03.**

**Exemple : Utilisation de `interpolate.lagrange`**

Calculons avec `interpolate.lagrange` de `scipy`, le polynôme de Lagrange de degré 2 qui passe par les points $(0, 0), (1, 1), (2, 2^3)$.

```
from scipy import interpolate
import numpy as np; import scipy as sp
X = np.array([0,1,2]); Y = X**3
P = sp.interpolate.lagrange(X,Y)
print(P) , P
        2
     3 x - 2 x    poly1d([ 3., -2.,  0.])
```

**Exercice 03**

**1.** Créer une fonction `BASE` d'argument la liste `Xabs` , qui renvoie la liste des polynômes de la base d'interpolation de Lagrange associée aux valeurs de la liste `Xabs`. Appliquer au cas où $Xabs = [0, 1, 2]$ et retrouver le résultat de l'exercice 02.

**2.** En utilisant `BASE` comme sous-procédure, créer une fonction `POL_LAG` d'arguments `Xabs` et `f` qui renvoie le polynôme d'interpolation de Lagrange passant par les points dont les abscisses sont les valeurs de $Xabs$ et les ordonnées sont les images par $f$ des valeurs de $Xabs$.
Appliquer avec $Xabs = [0, 1, 2]$ et la fonction $x \mapsto x^2$. Que remarque t-on ?
Appliquer avec $Xabs = [-1, 0, 1, 2, 3]$ et $f = \arctan$. On appellera dans la suite $P_1$ le dernier polynôme trouvé.

**3.** Tracer ensemble $P_1$ et la fonction arctan sur $[-1, 3]$. On pourra différencier les graphes avec l'option `linestyle`.

**4.** Retrouver $P_1$ en utilisant `interpolate.lagrange` de `scipy`.

***Indications :* 1.** *On pourra utiliser la fonction* `Polynomial` *de* `numpy.polynomial` *pour construire la liste des polynômes interpolateurs $L_0, ..., L_n$ de Lagrange. Au départ, on initialise avec* `L=[]` *et on prend* `n=len(Xabs)-1`*. Puis on construit les polynômes $L_0, L_1, ..., Ln$ dans une boucle double en faisant des produits de polynômes de degré 1. Pour éviter de faire appel à* `range(0,j)` *puis* `range(j+1,n+1)`*, on peut utiliser* `if i!=j` *pour raccourcir un peu. On initialise avec* `P=Polynomial([1])` *qui est $P = 1$. On rappelle enfin que* `Polynomial([a,b])` *est $a + bX$. Voir la* ***m thode 0.1*** *pour encore plus d'informations.*
**2.** *Les ordonnées sont les valeurs* `f(Xabs[i])` *pour $i$ dans le* `range(0,n+1)`*. De plus, la fonction* arctan *est prédefinie dans le module* `numpy` *d'alias* `np`*. On tapera donc* `np.arctan`
**3.** *Allez voir la* ***m thode 0.3****.*
**4.** *Allez voir la* ***m thode 0.2*** *et l'exemple développé juste après pour la syntaxe.*

## Exercice 04

Soit $n \in \mathbb{N}^\star$, on fixe $n+1$ points distincts $a \leqslant x_0 < x_1 < ... < x_{n-1} < x_n \leqslant b$.
Pour tout $j \in [\![0, n]\!]$, on note $L_j$ le $j^{\text{ème}}$ ***polynôme de la base d'interpolation de Lagrange*** associé à $(x_0, ..., x_n)$. En fixant encore $n+1$ points quelconques $y_0, ..., y_n$, on rappelle que le polynôme $L = \sum_{j=0}^{n} y_j L_j$ est le ***polynôme de Lagrange*** associé à $((x_0, y_0), ..., (x_n, y_n))$.

**1.** Créer en Python une fonction `Lagrange` qui a pour argument trois listes $X = [x_0, ..., x_n]$, $Y = [y_0, ..., y_n]$ et $T = [t_0, ..., t_p]$, où les $x_i$ sont deux à deux distincts et qui renvoie la liste $[L(t_0), ...L(t_p)]$, où $L$ est le polynôme de Lagrange plus haut.

**2.** Ici on suppose que $a = 0$, $b = 2$ et pour tout $j \in [\![0, n]\!]$, $x_j = j\dfrac{2}{n}$.
Tracer ensemble sur $[0, 2]$, le graphe de $f : x \mapsto e^{-x^2}$ et les polynômes de Lagrange associés au support $((x_0, f(x_0), ..., (x_n, f(x_n))$ pour tous les entiers $n$ appartenant à $[\![1, 19]\!]$.

***Indications :* 1.** *Ici* `Lagrange` *doit renvoyer une liste de flottants et ni un polynôme ni une fonction et donc* `Polynomial` *est inutile.*
**2.** *Pour tracer les polynômes de Lagrange, on peut s'inspirer de l'exercice 03 mais on va plutôt adapter la procédure* `Lagrange` *de la question précédente en la transformant en une fonction. Plus précisément, l'argument $T$ devient $x$ ce qui va permette de créer des fonctions polynomiales.*

**❑ M thode 0.3.— RAPPEL : Comment tracer plusieurs graphes ensemble avec Python**

On rentre préalablement `numpy` avec l'alias `np` et `matplotlib.pyplot` en alias `plt`.
Pour ***tracer deux graphes sur le même dessin***, $y = f(x)$ en rouge et $y = g(x)$ en vert pour $x \in [a, b]$, avec $p$ points, on écrit en suivant (avant de taper Entrée) :

```
X = np.linspace(a,b,p); plt.plot(X,f(X), color='r');
plt.plot(X,g(X), color='g'); plt.axis('equal'); plt.
    show()
# on peut rajouter :
plt.legend(('y = f(x)', 'y = g(x)'), 'best')
```

*Exemple : Tracer* sin *(en pointillé) et* cos *(en ligne discontinue) ensemble.*

```
import matplotlib.pyplot as plt ; import numpy as np ;
X = np.linspace(0,2*np.pi,500) ;
plt.plot(X,np.sin(X),linestyle='- -')
plt.plot(X,np.cos(X),linestyle=' :')
plt.legend(("y  = sin x ", " y = cos x ),'best '); plt.show( )
```