DS 7 – Maths & Info

Partie Informatique – Corrigé

Dans ce devoir, on modélise un polynôme $\sum_{k=0}^{n} a_k X^k$ par la liste $[a_0, a_1, \ldots, a_n]$ (dans cet ordre !)

Ainsi le polynôme $1 + 3X + X^4$ est modélisé par la liste $L = [1, 3, 0, 0, 4]$ et inversement la liste $L = [0, 2, 1, 3]$ correspond au polynôme $2X + X^2 + 3X^3$.

1. Un polynôme de degré n s'écrit $\sum_{k=0}^{n} a_k X^k$, où $a_0, \ldots, a_n$ sont $n+1$ réels, et $a_n \neq 0$, dont il faut une liste de longueur $\boxed{n+1}$ pour le modéliser.
2. La liste correspondant au polynôme $P = X^2 - 1$ est $\boxed{[-1, 0, 1]}$.
   Les polynômes correspondant aux listes $[1, 3, 0, 5]$ et $[2, 1, -1, 0, 0]$ sont respectivement $5X^3 + 3X + 1$ et $-X^2 + X + 2$.

Partie I – Racines

3. La fonction Python suivante qui prend en argument une **liste** correspondant à un polynôme P et un flottant a, et qui renvoie la valeur de $P(a)$.

```
def eval(P, a):
    s = 0
    for k in range(len(P)):
        s = s + P[k] * a**k
    return s
```

4. On sait que si $P = \sum_{k=0}^{n} a_k X^k$, alors $P' = \sum_{k=1}^{n} k a_k X^{k-1} = \sum_{k=0}^{n-1} (k+1) a_{k+1} X^k$. D'où la fonction `dérive(P)` (deux possibilités) :

```
def dérive(P):
    Pprime = []
    for k in range(1, len(P)):
        Pprime.append(k * P[k])
    return Pprime
```

```
def dérive(P):
    Pprime = []
    for k in range((len(P)-1):
        Pprime.append((k+1) * P[k+1])
    return Pprime
```

5. On en déduit une fonction Python `multiplicité(P, a)`, qui calcule l'ordre de multiplicité de a en tant que racine de P en calculant les valeurs successives de $P^{(k)}(a)$ jusqu'à obtenir un résultat non nul.

```
def multiplicité(P, a):
    m = 0
    polynome = P
        while eval(polynome, a) == 0:
            m = m + 1
            polynome = derive(polynome)
    return m
```

6. On procède par **dichotomie** pour obtenir une valeur approchée de la racine, de la manière suivante :

```
def cherche_racine(P, a, b, eps):
    while b - a > eps:
        Pa = eval(P, a)
        m = (a + b) / 2
        Pm = eval(P, m)
        if Pa * Pm <= 0 :
            b = m
        else :
            a = m
    return a
```

Partie II – Opérations sur les polynômes

7. (a) On considère les exemples de départ de l'énoncé.
   - Le polynôme $1 + 3X + X^4$ est modélisé par la liste $[1, 3, 0, 0, 4]$.
     Or $(1 + 3X + X^4) \times X = X + 3X^2 + X^5$ est modélisé par la liste $[0, 1, 3, 0, 0, 4]$.
   - Le polynôme $2X + X^2 + 3X^3$ est représenté par $[0, 2, 1, 3]$.
     Or $(2X + X^2 + 3X^3)X = 2X^2 + X^3 + 3X^4$, qui est représenté par $[0, 0, 2, 1, 3]$.
   - Dans ces deux exemples, on ajoute simplement un 0 au début de la liste.

   En effet, la multiplication par X "décale" les monômes, en transformant a_kX^k en a_kX^{k+1}. Ainsi, la multiplication par X dans les polynômes correspond à **l'ajout d'un** 0 **au début de la liste** en Python.

   (b) On en déduit la fonction `multiplication_par_X(P)` suivante :

```
def multiplication_par_X(P):
    return [0] + P
```

8. On en déduit la fonction Python `mult_monome(P, k)`, qui réalise k fois l'opération de multiplication par X :

```
def mult_monome(P, k):
    R = P
    for i in range(k):
        R = multiplication_par_X(P)
    return R
```

9. On considère la fonctions suivante sur les polynômes :

```
def mystere(P, c):
    R = []
    for e in P:
        R.append(e * c)
    return R
```

Cette fonction multiplie chacun des coefficients par le même réel c, donc il s'agit de la multiplication par un scalaire. Par exemple, `mystere([0,2,1,3], -2)` renverra $[0, -4, -2, -6]$, donc le polynôme $P = 2X + X^2 + 3X^3$ est transformé en $-4X - 2X^2 - 6X^3 = -2P$.

10. (a) Pour écrire la fonction `astuce(P1, P2)`, on commence par obtenir les longueurs des listes associées à $P1$ et $P2$. On rajoute des 0 à $P1$ tant qu'il est plus court que $P2$, et réciproquement. À la fin de l'exécution, les deux listes auront donc la même taille.

```python
def astuce(P1, P2):
    while len(P1) < len(P2): # si P1 est plus long, ça ne fait rien !
        P1.append(0)
    while len(P2) < len(P1): # si P2 est plus long, ça ne fait rien !
        P2.append(0)
    return P1, P2
```

(b) On en déduit la fonction `somme(P1, P2)`, qui commence par mettre $P1$ et $P2$ à la même taille, avant de faire la somme coefficient par coefficient.

```python
def somme(P1, P2):
    P1, P2 = astuce(P1, P2)
    S = []
    for k in range(len(P1)):
        S.append(P1[k] + P2[k])
    return S
```

11. **Bonus :** On remarque que si $Q = \sum_{k=0}^{n} a_k X^k$, alors $P \times Q = \sum_{k=0}^{n} a_k P X^k$. On part donc du polynôme nul (représenté par [0]), et on ajoute successivement chacun des polynômes $a_k X^k P$.

```python
def produit(P,Q):
    S = [0]
    for k in range(len(Q)):
        R = multiplie_monome(P, k)
        R = mystere(R, Q[k])
        S = somme(S, R)
    return S
```