Code de partage avec Capytale : 2c53-1072942

1 Introduction

Si P est un polynôme à coefficients réels de degré n, on peut l'écrire

$$P(x) = a_0 + a_1 x + \cdots + a_n x^n$$

Le polynôme P est entièrement déterminé par le $(n+1)$-uplet $(a_0, a_1, ..., a_n)$ de ses coefficients. Ainsi, on code le polynôme P par le vecteur-ligne (une liste de nombres en ligne) P formé de la suite de ses coefficients (listés par ordre croissant des puissances correspondantes) :

`P=[a0, a1, ..., an]`

2 Exercices

Exercice 1 - quelques tests

1. (a) Quel polynôme est représenté par `P=[-1, 0, 0, 3, 2]` ?

   Il s'agit de $P(x) = -1 + 0 \times x + 0 \times x^2 + 3x^3 + 2x^4 = -1 + 3x^3 + 2x^4$.

   (b) Que renvoie Python si on demande `P[3]` ? `len(P)` ?

   Avec `P[3]` Python renvoie `ans 0`, il s'agit en effet du troisième élément de liste `P` (en l'occurrence le coefficient de degré 2).
   Avec `len(P)`, on obtient la longueur de la liste de nombres, donc 5 ici. Cela correspond au degré plus une unité.

   (c) Quel est le point de vigilance ?

   $P(3)$ (ou autre : $P[3]\dots$) ne correspond pas à la valeur du polynôme pour $x = 3$ mais à la valeur du $3^{\text{ème}}$ nombre de la liste, ici 0.

2. Comment représenter le polynôme $Q(x) = 2x^4 - 3x^2 + 1$? `Q=[1, 0, -3, 0, 2]`

Exercice 2 - calcul de valeurs de P

1. Ecrire une fonction qui prend en argument un « polynôme » `P` et qui renvoie son degré.

   Notre fonction ne fonctionne que quand il n'y a pas de 0 superflus. Pour contourner cet éventuel problème, il faut intégrer des conditions (si le dernier coefficient vaut 0, regarder l'avant dernier... une boucle while est indiquée, cf. ci-dessous).

```
def degre(P):
    return len(P)
    -1
```

```
# pour prendre en compte le cas avec des zéros à la fin
def deg(P):
    n=len(P)-1
    while P[n]==0:
        n=n-1
    return n
```

2. Compléter la fonction suivante pour qu'elle renvoie la valeur du polynôme P en x (i.e. $P(x)$).

```
def evalpoly(P,x):
    y=P[0]
    for i in range(1,
    len(P)):
        y=y+P[i]*x**i
    return y
```

Il faut calculer $\sum_{k=0}^{n} a_k x^k$. Avec `P(k)`, on obtient le $k^{\text{ième}}$ élément de la liste `P` qui correspond en fait à a_{k-1}. Donc on le multipliera par `x**(k-1)`.
Ici on fait le choix de parcourir la liste `P` avec une boucle `for` pour obtenir tous les coefficients et on complète au fur et à mesure la valeur de `y` qui sera la sortie de la fonction.

On peut le tester sur différents polynômes, par exemple `R=[2,-5,2,4,-4,1]` qui correspond au polynôme $R(x) = x^5 - 4x^4 + 4x^3 + 2x^2 - 5x + 2$.

3. Trouver d'éventuelles racines évidentes du polynôme Q de l'exercice 1, puis factoriser Q.

```
Q=[1, 0, -3, 0, 2]
# Recherche de racines "évidentes"
for i in range(-1000,1000):
    if evalpoly(Q,i)==0:
        print(i)
```

Pour rechercher des racines « évidentes », on peut utiliser une boucle `for` de $-1\,000$ à $1\,000$ par exemple. Avec Q, on trouve que -1 et 1 sont des racines donc $Q(x) = (x-1)(x+1)R(x)$ puis on peut trouver $R(x)$ en développant et par identification.

Exercice 3 - polynôme dérivé

Écrire une fonction `derivepoly` prenant pour argument un « polynôme » P et renvoyant le « polynôme » dérivé P'.

```
def derivepoly(P):
    return [i*P[i] for i in
    range(1,len(P))]
```

Il suffit de multiplier chaque coefficient par l'indice où il se trouve dans la liste et de décaler la liste, ce qui est fait ci-contre en commençant la liste avec l'ancien `P[1]*1`