Polynômes : comment les multiplier par un scalaire et les additionner

poly07-scal-add.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""
écriture d'un programme pour évaluer
des polynomes
+ fonction de multiplication d'un polynôme pas un scalaire
+ fonction d'addition de deux polynômes
"""
from math import *

def polyeval(x,a):
    """
    application de l'agorithme de Horner
    cf. http://fr.wikipedia.org/wiki/M%C3%A9thode_de_Ruffini-Horner
    """
    n = len(a) - 1 # n = ordre du polynôme
    p =0.
    for i in range(n,-1,-1):
        p = p*x +a[i]
    return p

def polyscal(s,a):
    """
    polynôme multiplié par un scalaire s
    """
    b = []
    for coef in a:
        b.append(coef*s)
    return b    # on retourne les coefficients multipliés par s

def polyadd(a,b):
    """
    Addition de deux polynômes de coefficients a et b
    """
    r = a[:]      # on travaille sur une copie de a pour ne pas le
modifier
    t = b[:]      # idem pour b
    g = []        # polynôme somme
    n1 = len(r)   # ordre du premier polynôme
    n2 = len(t)   # ordre du second polynôme
    if n1 > n2:   # premier polynôme de plus haut degré que le second
        for i in range(n1-n2):
            t.append(0)
```

```
    elif n1 < n2: # second polynôme de plus haut degré que le premier
         for i in range(n2-n1):
             r.append(0)
    # r et t ont à présent la même longueur
    for i in range(len(r)):
        g.append(r[i] + t[i])
    return g  # on retourne les coefficients additionnés dans la liste
g

# différents tests :
x = 2.
a = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
print(polyeval(x,a))

varx = 0.5
varcoef = [1., 2., 3., 4., 5., 6., 7., 8., 9., 10.]
print(polyeval(varx,varcoef))

for j in range(0,11,1):
    rep = sin(polyeval(float(j) * 0.1,varcoef))
    print(rep)

u = [1, 2, 4, 8, 16]
v = [1, 3, 9, 27, 81, 243, 729]
print(u,v,polyadd(u,v))
```

Dans le fonction polyadd ci-dessus, l'idée est de compléter les listes par autant de zéros qu'il faut pour que la liste la plus courte devienne aussi longue que l'autre. On travaille en fait sur des copies des listes pour ne pas modifier les données originales, car python passe les paramètres des fonctions par référence et pas par valeur (*cf.* ici ou là). Polyadd aurait pu être conçue d'une autre manière, par exemple en additionnant les termes tant qu'on est en dessous du degré maximum du polynôme de degré minimum ! et en complétant ensuite par les coefficients de degré supérieur du polynôme de degré maximum. Comme la proposition suivante d'un étudiant !

Il est temps de créer des graphes de fonctions polynomiales. Comment faire ? Quelles librairies utiliser ?

Un exemple de graphe a été montré dans l'introduction du cours !

Voici un programme simple pour une fonction particulière (une sinusoïde amortie) :

simple-plot-sinus-amorti.py

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""
graphe d'une sinusoïde amortie
```

```
"""

from pylab import *

def sin_amort(t):
    s1 = sin(pi * t)
    e1 = exp(- t /8.)
    return s1 * e1

xvals = arange(0., 40., 0.1)
plot(xvals, sin_amort(xvals))
show()
# le même genre de graphique, avec des symboles rouges et lignes noires
plot(xvals, sin_amort(xvals), 'ro', xvals, sin_amort(xvals), 'k')
show()
```

Il faut donc faire un programme similaire en introduisant la fonction nécessaire et en adaptant les appels.

Réponse à la page suivante !