Cours 4 Récursivité (12/12/2019)

1. **Triangle de Pascal :** Construire une fonction récursive **pascal(n,p)** qui renvoie le triangle de pascal de $n+1$ lignes et $p+1$ colonnes par récursivité en utilisant des listes et **la méthode extend** sur les listes.
   Voici le principe :

```
def pascal(n,p) :
    if n==0:
        return **** à compléter ****
    t=pascal(n-1,p)
    **** à compléter ****
    t.extend(newline)
    return t
```

Par exemple $pascal(7,4)$ doit renvoyer :

```
[[1, 0, 0, 0, 0],
 [1, 1, 0, 0, 0],
 [1, 2, 1, 0, 0],
 [1, 3, 3, 1, 0],
 [1, 4, 6, 4, 1],
 [1, 5, 10, 10, 5],
 [1, 6, 15, 20, 15],
 [1, 7, 21, 35, 35]]
```

Puis afficher le triangle sous la forme symétrique ci-contre en enlevant les 0 et en centrant le contenu de chaque ligne de manière à avoir la forme d'un arbre. Si la ligne a une longueur paire on rajoute au milieu le symbole "_" qui permet de faire une ligne de longueur impaire comme la ligne 6

```
1 6 15 2_0 15 6 1
```

La ligne 6 a pour longueur 17 : entre les nombres on place juste un espace, on a aussi placé des espaces avant de manière à centrer la ligne du triangle de pascal par rapport à la dernière ligne affichée (ici la ligne 13)

```
                          1
                        1 1
                      1 2 1
                    1 3 3 1
                  1 4 6 4 1
                1 5 10 10 5 1
              1 6 15 2_0 15 6 1
            1 7 21 35 35 21 7 1
          1 8 28 56 7_0 56 28 8 1
        1 9 36 84 126 126 84 36 9 1
      1 10 45 120 210 252 210 120 45 10 1
    1 11 55 165 330 462 462 330 165 55 11 1
  1 12 66 220 495 792 924 792 495 220 66 12 1
1 13 78 286 715 1287 1716 1716 1287 715 286 78 13 1
```

2. **Dichotomie :**
   (a) Ecrire une fonction récursive `def dicho(f,a,b,eps) :` qui calcule par dichotomie la racine de f entre $a<b$ sachant que f continue et $f(a)f(b)<0$ à une erreur *"eps"* près
   (b) Par exemple trouver avec elle les trois racines de $P(x)=x^3-3x-1$ à $eps=10^{-5}$ près.
   (c) On reprend l'exemple du DS 3 :

$$R=\begin{pmatrix}0&0&1\\1&0&3\\0&1&0\end{pmatrix}$$

Définir les matrices R et D diagonales telles que R semblable à D ainsi que la matrice de passage P
Calculer par la diagonalisation M^{10} et vérifier que l'erreur sur les coefficients est en pourcentage des plus grands coefficients de l'ordre de $eps\%$
Vérifier que l'erreur pour M^n est de l'ordre de $\frac{n}{10}eps\%$ pour $n\in\{20,50,100,1000\}$

```
 1--> import numpy as np
```
Triangle de Pascal et affichage comme un arbre :
```
 2--> def pascal(n,p) :
 3-->     if n==0:
 4-->         return [[1]+[0 for k in range(p)]]
 5-->     t=pascal(n-1,p)
 6-->     l=[1]+[t[-1][k]+t[-1][k-1] for k in range(1,p+1)]
 7-->     t.append(l)
 8-->     return t
 9--> def arbre(q) :
10-->     u=pascal(q,q)
11-->     arbre=[]
12-->     for k in range(q):
13-->         m=u[k]
14-->         mot=str(m[0])
15-->         k=1
16-->         while m[k]!=0 and k<len(m):
17-->             mot+=' '+str(m[k])
18-->             k=k+1
19-->         a=len(mot)
20-->         if a%2==0 :
21-->             b=mot[:a//2]+'_'+mot[a//2:]
22-->             mot=b
23-->         arbre.append(mot)
24-->     n=len(arbre[-1])  #  longueur de la ligne
25-->     for k in range(q-1) :
26-->         p=len(arbre[k])
27-->         arbre[k]=' '*((n-p)//2)+arbre[k]
28-->     for mot in arbre :
29-->         print(mot)
```

Dichotomie récursive

```
30--> def dicho(f,a,b,eps) :
31-->     if b-a < 2*eps :
32-->         return (a+b)/2
33-->     else :
34-->         m=(a+b)/2
35-->         if f(m)*f(a) > 0 :
36-->             return dicho(f,m,b,eps)
37-->         else :
38-->             return dicho(f,a,m,eps)
39--> def f(x) :
40-->     return x**3-3*x-1
```

Racines du polynôme caractéristique à eps près

```
41--> eps=10**(-6) #  Constante à faire varier
42--> rac=[dicho(f,-3+2*i,-1+2*i,eps) for i in range(3) ]
```

Initialisation des trois matrices

```
43--> import numpy.linalg as la # Pour l'inverse de la matrice P
44--> R=np.zeros((3,3))
45--> D=np.zeros((3,3))
46--> P=np.zeros((3,3))
47--> R[0,2]=1
48--> R[1,0]=1
49--> R[1,2]=3
50--> R[2,1]=1
51--> for i in range(3) :
52-->     a=rac[i]
53-->     D[i,i]=a
54-->     P[0,i]=1 ; P[1,i]=a**2 ; P[2,i]=a
55--> P_inv=la.inv(P)
```

Fin initialisation des matrices

```
56--> def puiss(R,n) :
57-->     # Calcul de R^n
58-->     U=np.eye(3)
59-->     for k in range(n) :
60-->         U=np.dot(U,R)
61-->     return U
62--> import numpy.linalg as la
63--> def erreur(n) :
64-->     #  Erreur en pourcentage du plus grand coefficient de la matrice
65-->     Rn_dia=np.dot(np.dot(P,D**n),P_inv)
66-->     Rn_ex=puiss(R,n)
67-->     Delta=np.abs(Rn_ex-Rn_dia)
68-->     e=max(max(Delta.tolist()))/max(max(Rn_ex.tolist()))
69-->     return e
70--> for n in 20,50,100,1000 :
71-->     print(erreur(n)/n)
```

On trouve en effet une erreur en pourcentage de l'ordre de n*eps/10