TP 7 : Numpy, Matplotlib, et des dessins récursifs

2 Quelques tracés

Exercice 1. Par exemple par compréhension :

```
def cercle(x,y,r):
    T=np.linspace(0,2*pi,100)
    plt.plot([x+r*cos(t) for t in T], [y+r*sin(t) for t in T])
```

Remarque : Numpy possède des versions *vectorielles* des fonctions usuelles, c'est-à-dire qu'on peut les appliquer à un tableau, elle renvoie un tableau de même taille où la fonction a été appliquée à toutes les entrées. Voici une autre version de la fonction précédente :

```
def cercle(x,y,r):
    T=np.linspace(0,2*pi,100)
    plt.plot(x+r*np.cos(T), y+r*np.sin(T))
```

Exercice 2.

- Pour `fig1`, si on trace le cercle de centre $(x, 0)$ et de rayon r, le cercle suivant sera centré en $(x + \frac{3}{2}r, 0)$ (et aura rayon $r/2$) :

```
def fig1(n):
    plt.axis('equal')
    x=0
    r=1
    for i in range(n):
        cercle(x,0,r)
        x+=1.5*r
        r=r/2
    plt.show()
```

- Pour `fig2`, numérotons les lignes de cercles de $i = 0$ à $n-1$, de bas en haut. L'ordonnée des centres est augmenté de $\sqrt{3} = 2\sin(\pi/3)$ entre deux lignes, et l'abscisse du premier cercle d'une ligne est augmentée de $1 = 2\cos(\pi/3)$. D'où le code :

```
def fig2(n):
    plt.axis('equal')
    y=0
    x=0
    for i in range(n):
        for j in range(n-i):
            cercle(x+2*j,y,1)
        x+=1
        y+=sqrt(3)
    plt.show()
```

3 Fractales en Python

3.1 Tapis de Sierpinski

Exercice 3. Si $n > 1$, on effectue 8 appels récursifs. Il est commode d'utiliser deux boucles `for` imbriquées de longueur 3, ce qui fait 9 passages dans la boucle interne, chacun correspondant à un petit carré. On exclut le carré central qui a déja été colorié en noir.

```
def tapis_sierpinski(n):
    plt.axis('equal')
    def aux(x,y,c,n): #travaille sur le carré [x,x+c]*[y,y+c]
        c2=c/3
        plt.fill([x+c2,x+c2,x+2*c2,x+2*c2],[y+c2,y+2*c2, y+2*c2, y+c2],"k") #carré central
        if n>0:
            for i in range(3):
                for j in range(3):
                    if i!=1 or j!=1:
                        aux(x+i*c2, y+j*c2,c2,n-1)
    aux(0,0,1,n)
    plt.show()
```

3.2 Triangle de Sierpinski

Pour le tapis de Sierpinski, l'idée est la même, à part qu'on fait seulement 3 appels récursifs. La fonction `aux` suivant prend 3 sommets d'un triangle en entrée, en plus de l'entier n. Ces sommets sont donnés sous forme de tableau Numpy, ce qui facilite grandement le calcul de milieux.

```
def triangle_sierpinski(n):
    plt.axis('equal')
    def aux(c1,c2,c3,n):
        m1, m2, m3 = (c2+c3)/2, (c1+c3)/2, (c2+c1)/2
        plt.fill([m1[0], m2[0], m3[0]], [m1[1], m2[1], m3[1]], "k")
        if n>0:
            aux(c1,m2,m3, n-1)
            aux(m1,c3,m2, n-1)
            aux(c2,m3,m1, n-1)
    aux(np.array([0.,0.]), np.array([1.,0.]), np.array([0.5,3**0.5/2]), n)
    plt.show()
```

3.3 Flocon de Von Koch

```
def flocon(n):
    plt.axis('equal')
    def koch(p,q,n):
        if n==0:
            plt.plot([p[0],q[0]], [p[1],q[1]],"k")
        else:
            u=(q-p)/3
            x,y=u
            c,s=cos(pi/3), sin(pi/3)
            v=np.array([c*x-s*y, s*x+c*y])
            koch(p,p+u,n-1)
            koch(p+u,p+u+v,n-1)
            koch(p+u+v,p+2*u,n-1)
            koch(p+2*u,q,n-1)
    p1,p2,p3 = np.array([0.,0.]), np.array([0.5,sqrt(3)/2]),np.array([1.,0.])
    koch(p1,p2,n)
    koch(p2,p3,n)
    koch(p3,p1,n)
    plt.show()
```

Explications : dans la figure ci-dessous, les points p' et q' se calculent facilement comme $\frac{2p+q}{3}$ et $\frac{p+2q}{3}$. Le vecteur $\overrightarrow{p'q'} = \frac{1}{3}\overrightarrow{pq}$ se calcule facilement également (pour obtenir les coordonnées du vecteur $\vec{AB}$, il suffit de soustraire les coordonnées de A à celles de B). Le vecteur $\overrightarrow{p'r}$ est obtenu par rotation d'angle $\pi/3$ de $\overrightarrow{p'q'}$. Pour obtenir les coordonnées de r, il suffit de rajouter les coordonnées de p' à $\overrightarrow{p'r}$. Essentiellement, le code de la fonction calcule ces points, et effectue 4 appels récursifs si $n > 0$, sinon il trace simplement le segment.