5bis Fractales

July 26, 2024

[1]:
```
import matplotlib.pyplot as plt
from math import sin, cos, pi
from random import random, gauss
import numpy as np
```

1 Triangle de Sierpinski

[2]:
```
def m(A, B):
    xA, yA = A
    xB, yB = B
    return (xA + xB)/2, (yA + yB)/2
```

[3]:
```
def trace_triangle(ax, T, n, lw):
    """
    Trace dans la zone ax le triangle de Sierpinski.
    """
    if n != 0:
        (xA, yA), (xB, yB), (xC, yC) = A, B, C = T
        ax.plot([xA, xB, xC, xA], [yA, yB, yC, yA], color='black', lw=lw)
        trace_triangle(ax, (m(A, B), A, m(A, C)), n-1, 0.7*lw)
        trace_triangle(ax, (m(B, C), B, m(B, A)), n-1, 0.7*lw)
        trace_triangle(ax, (m(C, A), C, m(C, B)), n-1, 0.7*lw)
```

[4]:
```
fig, ax = plt.subplots()
ax.axis('equal')
trace_triangle(ax, ((0, 0), (1, 0), (cos(pi/3), sin(pi/3))), 6, lw=4)
plt.show()
```

2 Flocon de Koch

[5]:
```
def rot(v, alpha):
    c = cos(alpha)
    s = sin(alpha)
    return np.array([[c, -s], [s, c]]) @ v
```

[6]:
```
def koch(A, B, n):
    """
    Retourne la liste des points (x, y) du flocon.
    """
    if n == 0:
        return []
    else:
        P = A + (1/3) * (B - A)
        R = A + (2/3) * (B - A)
        Q = P + rot((R - P), pi/3)
        return [A] + koch(A, P, n-1) + koch(P, Q, n-1) + koch(Q, R, n-1) +␣
↪koch(R, B, n-1) + [B]
```

[7]:
```
def zippe(li):
    """
    Retourne les deux listes de coordonnées à partir des deux listes des points.
    """
    if li == []:
        return [], []
    else:
        x, y = li[0]
        xx, yy = zippe(li[1:])
        return [x] + xx, [y] + yy
```

[8]:
```
import sys
sys.setrecursionlimit(4000) # nécessaire pour n=5
```

[9]:
```
fig, ax = plt.subplots()
ax.axis('equal')
ax.axis('off')
X, Y = zippe(koch(np.array([0, 0]), np.array([1, 0]), n=5))
ax.plot(X, Y, color='black')
plt.show()
```

3 Terrain fractal

[10]:
```
A = np.array([0, random()])
B = np.array([1, random()])
```

[11]:
```
def terrain(ax, A, B, k, n):
    if n == 0:
        xA, yA = A
        xB, yB = B
        ax.plot([xA, xB], [yA, yB], color='black')

    else:
        M = (A + B)/2
```

```
        M[1] += random()/k
        terrain(ax, A, M, 1.8*k, n-1)
        terrain(ax, M, B, 1.8*k, n-1)
```

[12]:
```
fig, ax = plt.subplots()
terrain(ax, A, B, 1, 6)
plt.show()
```

4 Arbre fractal

[13]:
```
def tree(ax, M0, M1, alpha1, alpha2, n):
    if n != 0:
        x0, y0 = M0
        x1, y1 = M1
        ax.plot([x0, x1], [y0, y1], color='black')
        M2 = M1 + rot(M1 - M0, alpha1)
        M3 = M1 + rot(M1 - M0, -alpha1)
        tree(ax, M1, M2, alpha1, alpha2, n-1)
        tree(ax, M1, M3, alpha1, alpha2, n-1)
```

[14]:
```
M0 = np.array([0, 0])
M1 = np.array([0, 1])
fig, ax = plt.subplots()
tree(ax, M0, M1, pi/12, pi/12, 5)
plt.show()
```

[15]:
```
def tree2(ax, M0, M1, alpha1, alpha2, lw, n):
    if n != 0:
        x0, y0 = M0
        x1, y1 = M1
        ax.plot([x0, x1], [y0, y1], color='black', lw=lw)
        M2 = M1 + 0.7*rot(M1 - M0, alpha1)
        M3 = M1 + 0.7*rot(M1 - M0, -alpha1)
        tree2(ax, M1, M2, alpha1, alpha2, lw/1.5, n-1)
        tree2(ax, M1, M3, alpha1, alpha2, lw/1.5, n-1)
```

[16]:
```
M0 = np.array([0, 0])
M1 = np.array([0, 1])
fig, ax = plt.subplots()
tree2(ax, M0, M1, pi/7, pi/7, 8, 9)
plt.show()
```

[17]:
```
def tree3(ax, M0, M1, alpha1, alpha2, k1, k2, lw, n):
    if n != 0:
        x0, y0 = M0
        x1, y1 = M1
        ax.plot([x0, x1], [y0, y1], color='black', lw=lw)
```

```
        M2 = M1 + k1*rot(M1 - M0, alpha1)
        M3 = M1 + k2*rot(M1 - M0, -alpha1)

        alpha1 = gauss(pi/8, pi/16)
        alpha2 = gauss(pi/8, pi/16)
        k1 = random()**0.4
        k2 = random()**0.4

        tree3(ax, M1, M2, alpha1, alpha2, k1, k2, lw/1.5, n-1)
        tree3(ax, M1, M3, alpha1, alpha2, k1, k2, lw/1.5, n-1)
```

[18]:
```
M0 = np.array([0, 0])
M1 = np.array([0, 1])
fig, ax = plt.subplots()
tree3(ax, M0, M1, pi/7, pi/7, 0.7, 0.7, 8, 9)
plt.show()
```