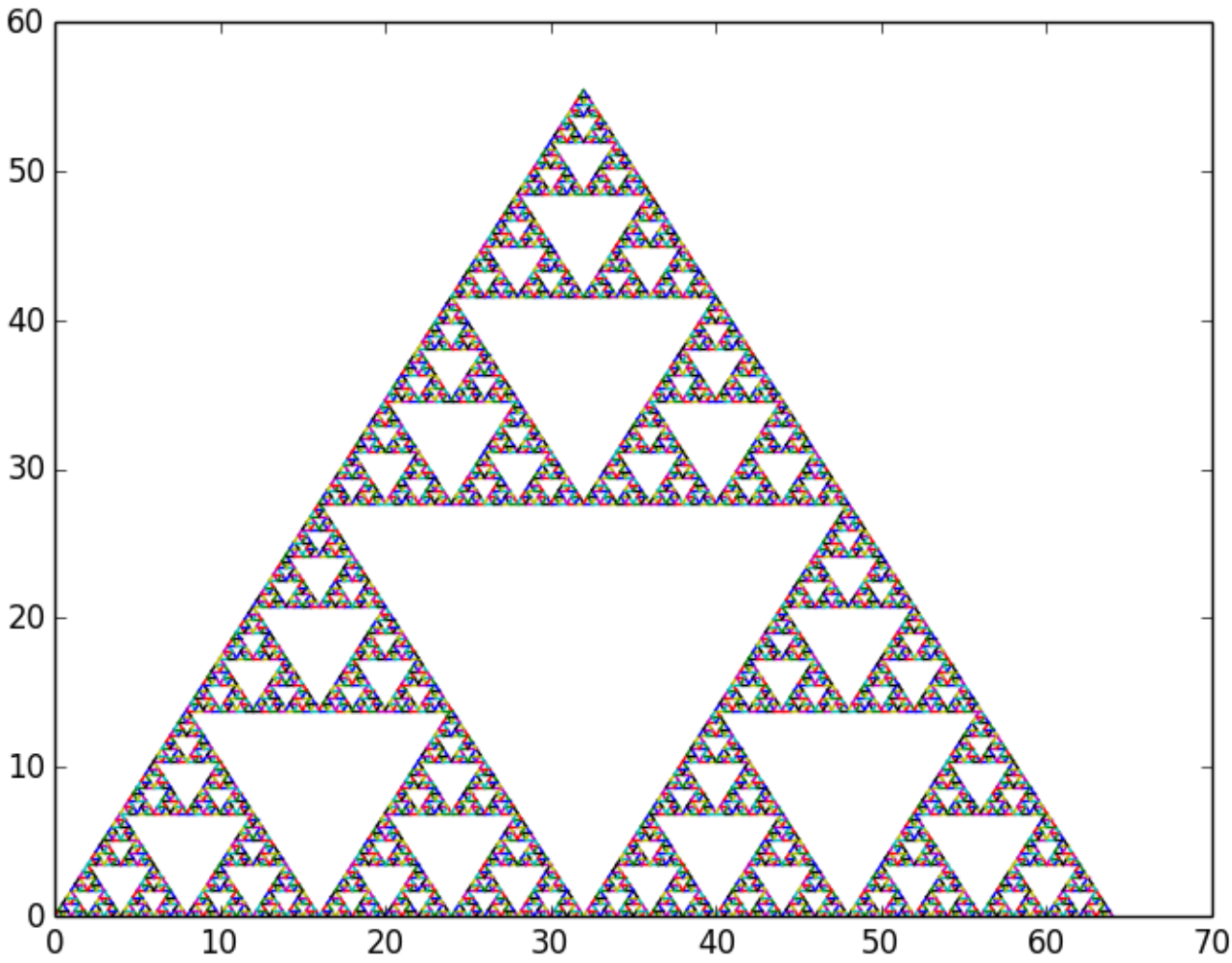
Récursivité et graphiques

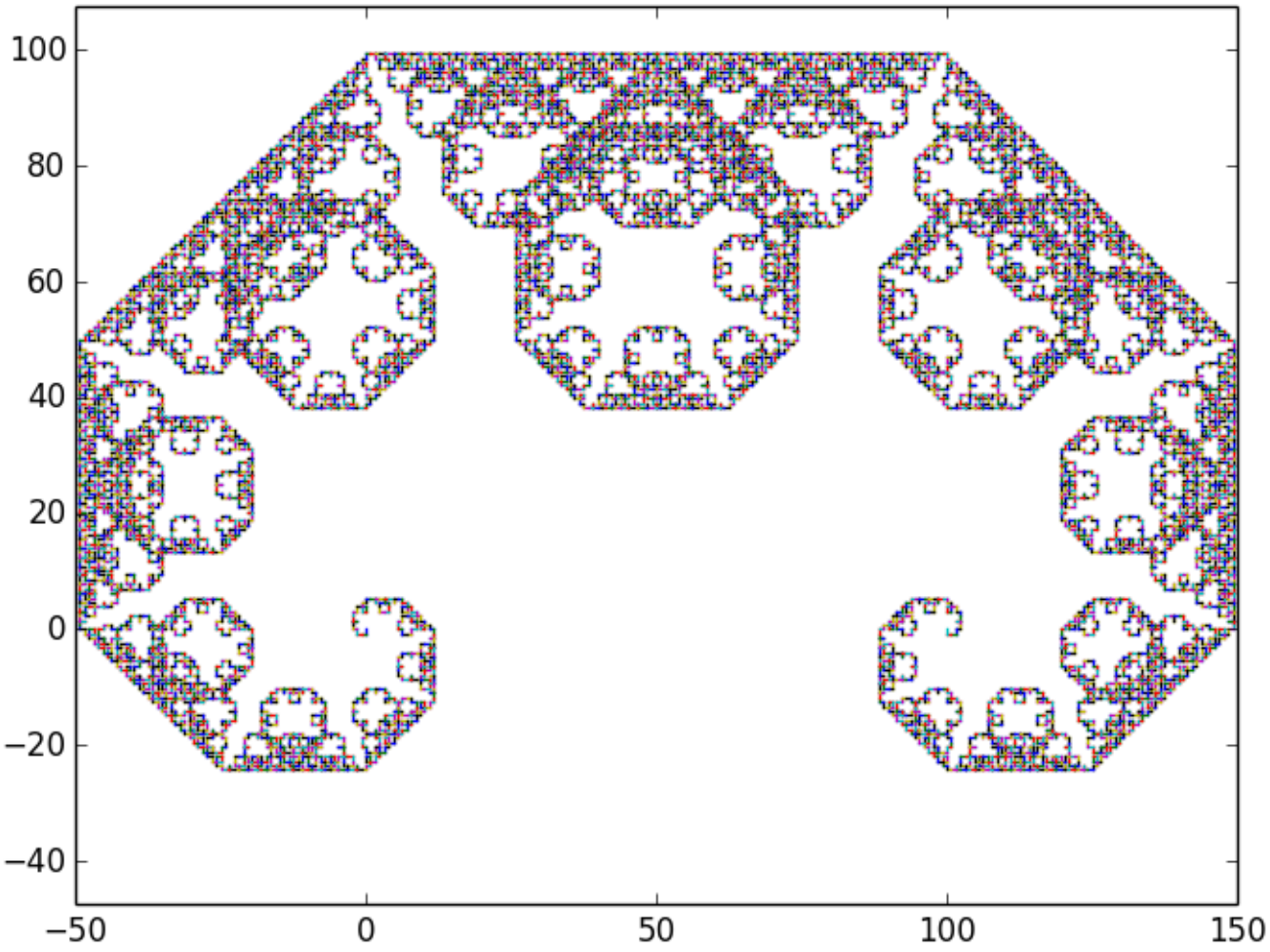
1 Le triangle de Sierpinski



Programme

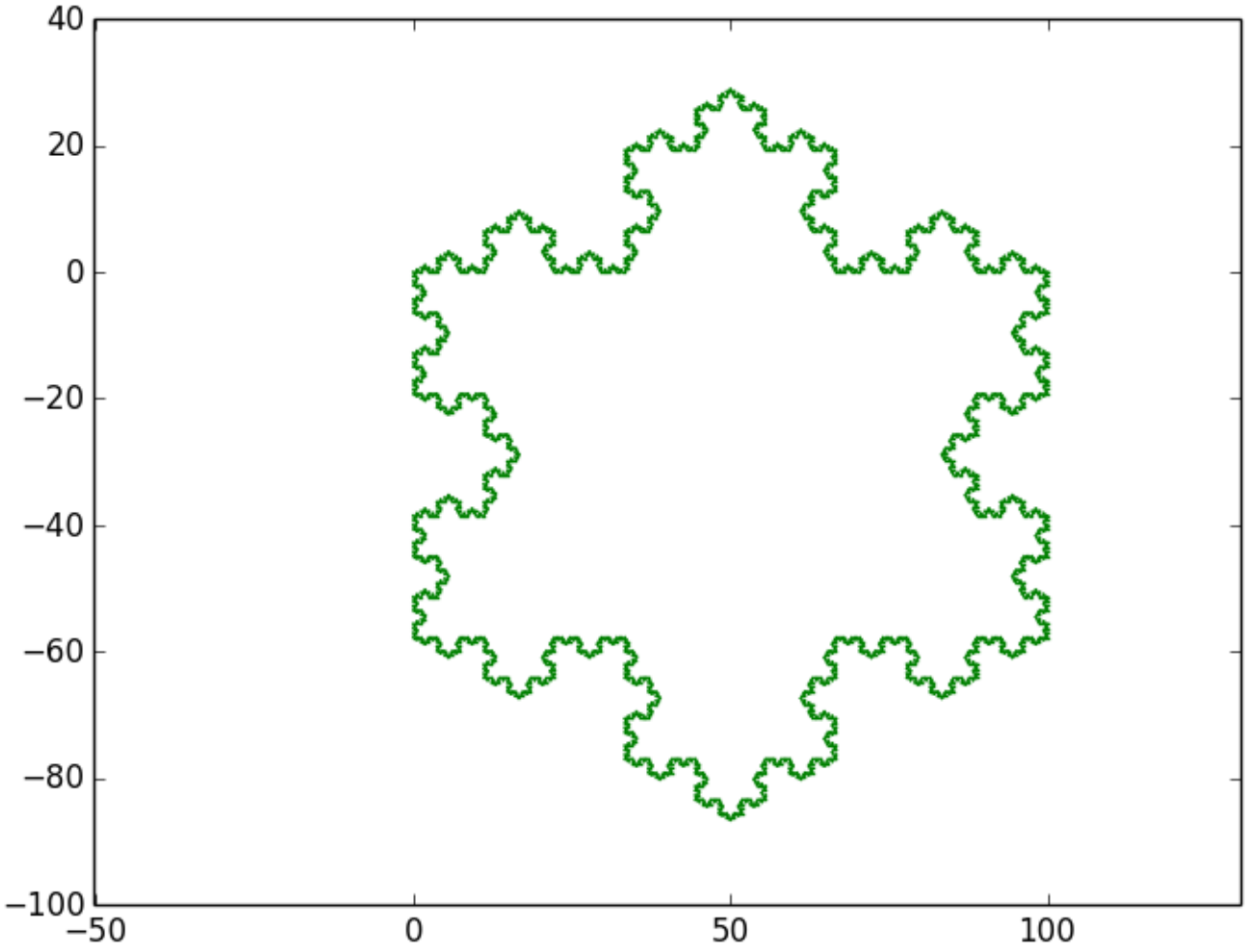
```python
import matplotlib.pyplot as plt
from math import sqrt
rac3 = sqrt(3)
r_min = 1

# trace le triangle de côté r
# dont le sommet en bas à gauche est (x, y)

def triangle(x, y, r):
    if r > r_min:
        triangle(x, y, r/2)
        triangle(x + r/2, y, r/2)
        triangle(x + r/4, y + r*rac3/4, r/2)
    else:
        plt.plot([x], [y], marker = '.')

triangle(0, 0, 100)
```

2 Le crabe



```
import matplotlib.pyplot as plt
from math import *

r_min = 1
rac2 = sqrt(2)

# trace le crabe d'origine (x, y), d'envergure r,
# incliné de l'angle t par rapport à l'horizontale.

def crabe(x, y, r, t):
    if r > r_min:
        r1, t1, t2 = r/rac2, t + pi/4, t - pi/4
        crabe(x, y, r1, t1)
        crabe(x + r1*cos(t1), y + r1*sin(t1), r1, t2)
    else:
        x1 = x + r*cos(t)
        y1 = y + r*sin(t)
        plt.plot([x,x1],[y,y1])

crabe(0, 0, 100, 0)
plt.axis('equal')
```

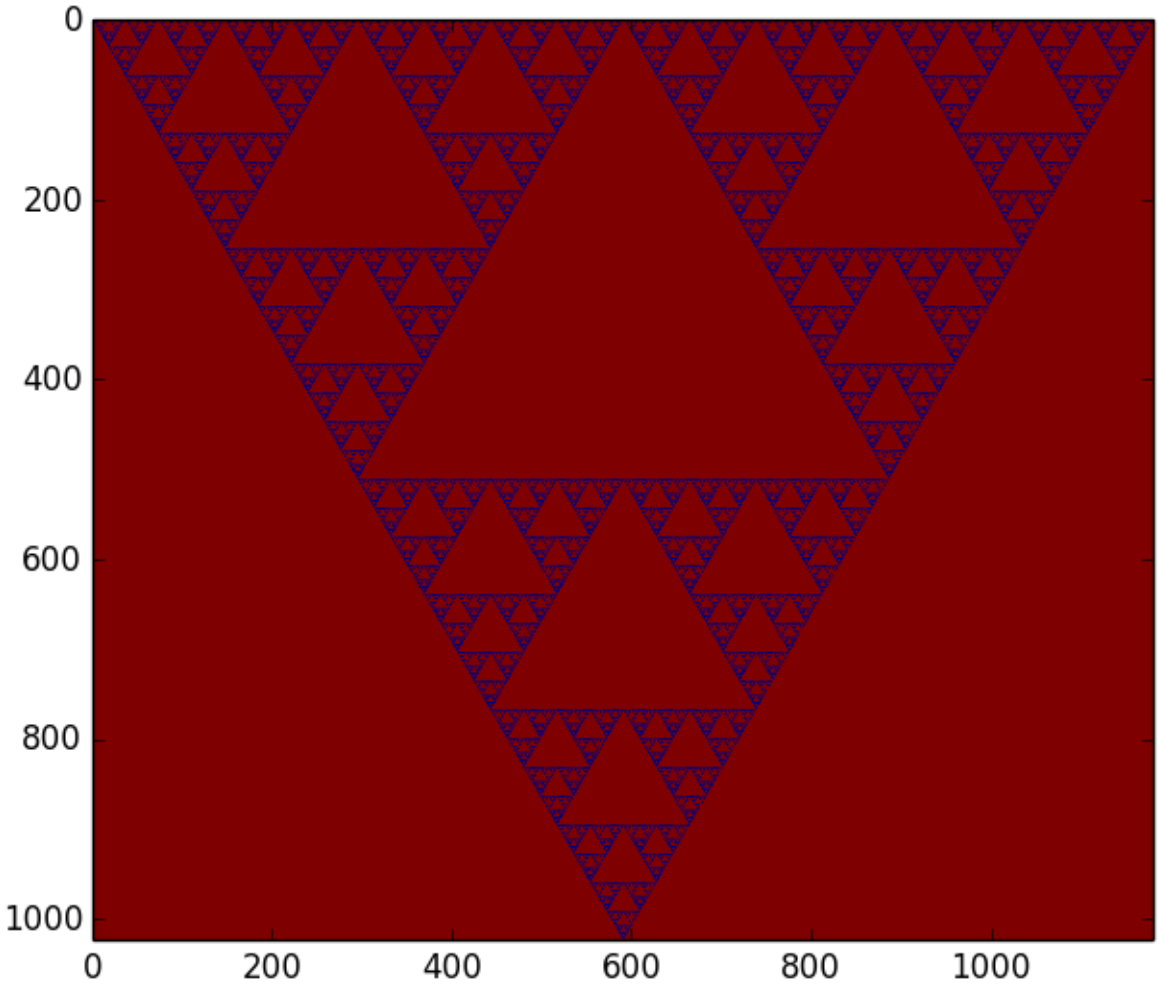
3 Le flocon de Koch



Programme

```
import matplotlib.pyplot as plt
from math import *
R, r_min = 30, 1

# flocon1(x, y, r, t) trace à partir du point (x, y),
# sur une distance r, dans la direction définie par l'angle t.

def flocon1(x, y, r, t):
    if r > r_min:
        r1, t1, t2 = r/3, t + pi/3, t - pi/3
        flocon1(x, y, r1, t)
        x1, y1 = x + r1*cos(t), y + r1*sin(t)
        flocon1(x1, y1, r1, t1)
        x2, y2 = x1 + r1*cos(t1), y1 + r1*sin(t1)
        flocon1(x2, y2, r1, t2)
        x3, y3 = x + 2*r1*cos(t), y + 2*r1*sin(t)
        flocon1(x3, y3, r1, t)
    else:
        plt.plot([x, x + r*cos(t)], [y, y + r*sin(t)], 'g')
plt.axis('equal')
flocon1(0, 0, R, 0)
flocon1(R, 0, R, -2*pi/3)
flocon1(R/2, - R*sin(pi/3), R, 2*pi/3)

plt.pause(1)

# flocon2(x, y, t, r) trace à partir du point (x, y),
# dans la direction définie par l'angle t,
# sur une distance r, et renvoie le point d'arrivée.

def flocon2(x, y, t, r):
    if r < r_min:
        x1 = x + r*cos(t)
        y1 = y + r*sin(t)
        plt.plot([x, x1],[y, y1],'r')
        return x1, y1
    else:
        x1, y1 = flocon2(x, y, t, r/3)
        x2, y2 = flocon2(x1, y1, t + pi/3, r/3)
        x3, y3 = flocon2(x2, y2, t - pi/3, r/3)
        x4, y4 = flocon2(x3, y3, t, r/3)
        return x4, y4

plt.axis('equal')
u1, v1 = flocon2(0, 0, 0, R)
u2, v2 = flocon2(u1, v1, -2*pi/3, R)
u3, v3 = flocon2(u2, v2, 2*pi/3, R)
```

4 Le triangle de Sierpinski, version matricielle



Programme

```python
import matplotlib.pyplot as plt
import numpy as np
rac3 = np.sqrt(3)
n = 8
H = 2**n
L = int(H*2 / rac3)
mat = np.ones((H, L))

for i in range(H):
    mat[i][int(i / rac3): int(L - i / rac3)] =  0

def blanc(i, j, h):
    for k in range(int(h)):
        mat[i + k][int(j - k/rac3): int(j + k/rac3)] =  1


def spk(i1, i2, j):
    if i2 <= 3 + i1:
        return
    mi = (i1 + i2)/2
    mj = j + (i2 - i1)/rac3
    spk(i1, mi, j)
    spk(i1, mi, mj )
    spk(mi, i2, (j + mj)/2)
    blanc(i1, mj, (i2 - i1)/2)

spk(0, H, 0)
plt.imshow(mat)

# Dans spk, (i1, j) sont les coordonnées du sommet en haut à gauche.
# (i1, i2) sont les lignes limites du triangle.
# Dans blanc, (i, j) sont les coordonnées du sommet en haut
# et h est la hauteur du triangle.
```