

什么是Julia分形集?

@20241024_085033

user: 什么是Julia分形集?

[Timestamp: 2024/10/24 08:50:33]

ai: Julia分形集是一种在数学中广泛研究的分形结构，以法国数学家皮埃尔-西蒙·Julia命名。它是在复平面上定义的一组点，这些点的迭代过程通过特定的复数函数（通常是二次多项式）进行。对于每一个复数 c ，都可以定义一个Julia集 J_c ，它是所有发散到无限的点的集合，其中的点按照该复数函数的迭代不断远离复平面上的某个固定点。

Julia分形集与 Mandelbrot 集合有着密切的联系。

Mandelbrot 集合是所有 Julia 集合 J_c 的边界。换句话说，如果 c 是Mandelbrot集合上的一个点，则对应的Julia集 J_c 是连通的；如果 c 不在Mandelbrot集合上，则 J_c 是非连通的。

Julia分形集因其独特的形态和数学性质而被广泛研究和绘制。它们通常呈现出极其复杂的自相似结构，即在放大后可以发现与原始图像相似的部分。这些分形结构不仅在数学领域具有重要意义，也因其艺术价值而在更广泛的公众中受到欢迎。

[Timestamp: 2024/10/24 08:51:34]

user: 常见的绘制Julia集的算法有哪些？如何用代码实现？

[Timestamp: 2024/10/24 08:52:08]

ai: 绘制Julia集的两种常见算法是逃逸时间算法（Escape Time Algorithm）和判别线段法（Boundary colloidal

method)。下面我将分别介绍这两种方法，并给出使用 Python 编程语言的一个简单实现示例。

1. 逃逸时间算法

逃逸时间算法基于一个点是否发散到无穷远来确定它是否属于 Julia 集。对于给定的复数 c 和一个起始点 z_0 ，我们计算迭代序列 $z_{n+1} = z_n^2 + c$ 。如果这个序列中的某个点 z_n 超过某个发散半径（例如，100），则认为该点不在 Julia 集内，并记录它需要多少次迭代才能逃逸到无穷远。

```

import numpy as np
import matplotlib.pyplot as plt

def mandelbrot Julia(c, max_iter=100):
    z = c
    for n in range(max_iter):
        if abs(z) > 2:
            return n
        z = z**2 + c
    return max_iter

def plot_julia集(c, width=800,
height=800, real_min=-2, real_max=1,
imag_min=-1.5, imag_max=1.5):
    r = real_max - real_min
    i_r = imag_max - imag_min
    x, y = np.linspace(real_min,
real_max, width),
np.linspace(imag_min, imag_max,
height)
    grid = np.array([x + 1j * y[i] for
i in range(height)])
    colors = np.array([mandelbrot
Julia(c, max_iter=255) for z in grid],
dtype=int)
    plt.imshow(colors.T, extent=
(real_min, real_max, imag_min,

```

```
imag_max), cmap='hot')
plt aspect('equal')
plt.colorbar()
plt.show()

# 使用 example
c = -0.7 + 0.27015j
plot_julia集(c)
```

2. 判别线段法

判别线段法是基于Julia集与 Fatou 集（非Julia集）的边界在复平面上的迭代特性。这种方法通过计算点在迭代过程中的轨道，来判断它是否属于 Julia 集。

```
def mandelbrot Fatou(c, max_iter=100):
    z = c
    for n in range(max_iter):
        if abs(z) > 2:
            return n
        z = z**2 + c
    return max_iter
```

```
def plot_julia集(c, width=800,
height=800, real_min=-2, real_max=1,
imag_min=-1.5, imag_max=1.5):
    r = real_max - real_min
    i_r = imag_max - imag_min
    x, y = np.linspace(real_min,
real_max, width),
np.linspace(imag_min, imag_max,
height)
    grid = np.array([x + 1j * y[i] for
i in range(height)])
    colors = np.array([mandelbrot
Fatou(c, max_iter=255) for z in grid],
dtype=int)
    plt.imshow(colors.T, extent=
(real_min, real_max, imag_min,
imag_max), cmap='hot')
    plt aspect('equal')
    plt colorbar()
```

```
plt.show()

# 使用 example
c = -0.7 + 0.27015j
plot_julia集(c)
```

这两种方法都可以用来绘制Julia集，其中逃逸时间算法通常更快，但判别线段法在某些情况下可能提供更详细和精确的图像。

[Timestamp: 2024/10/24 08:54:57]

user: 现在我希望借助Geogebra来实现可能吗？例如我考察一个矩形区域中的点，根据它是否属于Julia集进行彩色，进而得到Julia分形集

[Timestamp: 2024/10/24 08:56:24]

ai: Certainly! Geogebra is a powerful tool for visualizing mathematical concepts, including Julia sets. Below, I'll guide you through the steps to create a Julia set visualization in Geogebra.

Step-by-Step Guide to Creating a Julia Set in Geogebra

1. Define the Complex Constant c :

- Open Geogebra and define your complex constant c using the **ComplexNumber** command.

$$c = -0.7 + 0.27015i$$

2. Define the Function $f(z) = z^2 + c$:

- Geogebra doesn't directly support complex functions, but you can use the **ComplexNumber** command to define the function in terms of real and imaginary parts.

```
z = x + yi  
f(z) = (z^2 + c).real() + (z^2 +  
c).imag() i
```

3. Define a Function to Check Escape:

- Create a function that checks if a point escapes the Julia set based on a maximum number of iterations.

```
Julia(z, maxIter) = If( abs(z) > 2,  
maxIter, 1 + Julia(f(z), maxIter -  
1))
```

4. Create a Grid of Points:

- Define the real and imaginary ranges to create a grid of points.

```
realMin = -2
realMax = 1
imagMin = -1.5
imagMax = 1.5
width = 500
height = 500

realStep = (realMax - realMin) /
width
imagStep = (imagMax - imagMin) /
height

points = Sequence(Sequence(x + y i,
x, realMin, realMax, realStep), y,
imagMin, imagMax, imagStep)
```

5. Apply the Julia Function to Each Point:

- Apply the **Julia** function to each point in the grid.


```
escapeTimes = Sequence(Julia(p,  
100), p, points)
```

6. Create a Color List for the Points:

- Define a color list based on the escape times.

```
colors = If(escapeTimes > 50,  
"Red", If(escapeTimes > 25,  
"Yellow", If(escapeTimes > 10,  
"Green", "Blue"))))
```

7. Plot the Points with Colors:

- Use the **Sequence** command to plot each point with its corresponding color.

```
Sequence(If( colors[i] == "Red",  
Point(i + 1), If( colors[i] ==  
"Yellow", Point(i + 1), If(  
colors[i] == "Green", Point(i + 1),  
Point(i + 1)))),  
i, 1, length(colors))
```

8. **Adjust the View:**

- Adjust the axes and the view to properly display the Julia set.

```
SetView(-2, 1, -1.5, 1.5)
```

By following these steps, you can create a basic visualization of a Julia set in Geogebra. You can experiment with different values of c and adjust the number of iterations to explore different parts of the Julia set.

Here is a simplified version of the script for your convenience:

```
c = -0.7 + 0.27015i
```

```
width = 500
```

```
height = 500
```

```
realMin = -2
```

```
realMax = 1
```

```
imagMin = -1.5
```

```
imagMax = 1.5
```

```
realStep = (realMax - realMin) / width
```

```
imagStep = (imagMax - imagMin) /
```

```
height
```

```
points = Sequence(Sequence(x + y i, x,  
realMin, realMax, realStep), y,  
imagMin, imagMax, imagStep)
```

```
escapeTimes = Sequence(Julia(p, 100),  
p, points)
```

```
colors = If(escapeTimes > 50, "Red",  
If(escapeTimes > 25, "Yellow",  
If(escapeTimes > 10, "Green",  
"Blue"))))
```

```
Sequence(If( colors[i] == "Red",  
Point(i + 1), If( colors[i] ==  
"Yellow", Point(i + 1), If( colors[i]  
== "Green", Point(i + 1), Point(i +
```

```
1))))),  
      i, 1, length(colors))  
  
SetView(-2, 1, -1.5, 1.5)
```

This script should run in Geogebra and produce a basic Julia set visualization. You can further enhance it by adding more colors or adjusting the iteration count for better detail.

[Timestamp: 2024/10/24 09:00:12]