

# 利用辐角原理求解复变函数零点

学院：少年班学院 学号：PB21000184 姓名：苗桐畅

2023 年 6 月 21 日

## 1 科学技术原理

**定理 1.1** 设  $f(z)$  是单连通区域  $D \subset \mathbb{C}$  上的全纯函数，简单闭曲线  $\gamma \subset D$ ， $f(z) \neq 0$  as  $z \in \gamma$ ，设点  $z$  沿  $\gamma$  顺时针移动一周时，点  $f(z)$  绕原点顺时针转动的圈数为  $l(\gamma, f)$ ，称为卷绕数。若  $l(\gamma, f) \neq 0$ ，则  $f(z)$  在  $\gamma$  的内部有零点。

上述定理被称为辐角原理，证明细节不再赘述。例 1.1 是该定理的一个直观展示。

**例 1.1** 设  $f(z) = (z + \frac{1}{2})(z + \frac{1}{3})(z + \frac{1}{4})$ ，曲线  $\gamma: |z| = 1$ ，如下图：

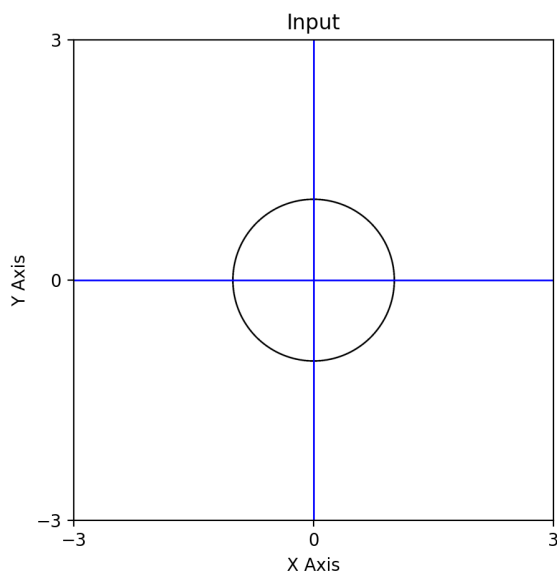


图 1:  $z \in \gamma$  轨迹

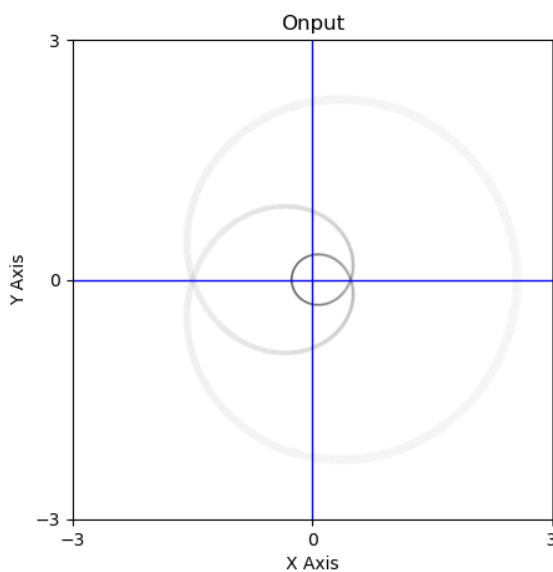


图 2:  $f(z)$  轨迹

可以看出点  $f(z)$  绕着原点转了三圈，这意味着  $f(z)$  在  $\gamma$  内部，即单位圆盘上有三个零点。这三个零点就是  $-\frac{1}{2}, -\frac{1}{3}, -\frac{1}{4}$ 。

注：本例中的图片由 test.py 生成。

根据这一定理，可以快速判断给定区域（一般选择矩形区域）内是否存在零点，之后将其四等分为四个更小的矩形区域，分别求卷绕数，进一步缩小零点所在的范围。这便是二分法的延伸：四分法。

## 2 设计方案

本程序主要部分有两个：Pic.py 和 Calculate.py。（test.py 仅仅用在了例 1.1 当中）

## 2.1 可视化

Pic.py 的功能是实现复变函数的可视化，能够快速但粗糙地展示一个复变函数的图像，让使用者能够判断零点所在大致区域。程序结构较为简单：

- (1) colorf: 输入一个复数  $z_0$ ，计算  $f(z_0)$  并输出一个 RGB 值， $f(z_0)$  的辐角决定了其颜色，模长决定了其明暗程度，模长越小点越明亮。
- (2) paintall: 输入一个矩形区域，遍历区域内所有的采样点并调用 colorf 得到 RGB 值，存储在三维数组 Picmat 中，利用 matplotlib.pyplot 的绘图功能进行绘图。

## 2.2 算法

Calculate.py 的主要功能是计算给定区域内函数的零点，次要功能是给出一张寻找零点的轨迹图。它主要包含下列要素：

- (1) circ: 输入一系列辐角值，返回卷绕数。
- (2) sol: 输入一个矩形区域，计算从左下角开始顺时针一周的辐角值，调用 circ 函数计算卷绕数并返回。
- (3) find: 输入一个矩形区域，和当前区域卷绕数。该函数是递归时的主干函数，无返回值，搜索范围的面积小到一定程度（由参量 prec 控制）时当前递归结束，并把区域中心点加入到全局变量 Pointlist 中，代表找到了一个目标点。
- (4) 主函数：整理、判断和输出得到的点列。

程序开始时建立一张画布，搜寻零点的过程中会把计算过的点绘制到画布上，最后得到一张寻找零点的轨迹图。可以与 Pic.py 绘制出的粗略图进行对比，判断是否有遗漏的零点，然后重新选择区域并计算。

## 3 创新性描述

卷绕数是一种在流体力学和拓扑学中广泛应用的概念，而将其应用于求解二维方程（等价于求复变函数零点，详见例 4.1）则是一个比较新颖的思路。利用卷绕数给出的四分法是一维方程二分法的合理创新和延伸。

此外我还对算法做出了一些改进。如果直接应用四分法求解零点，会有一个缺陷：实际上，如果函数在区域内含有极点（不全纯），则卷绕数的绝对值等于区间内零点的个数和极点的个数之差，这说明如果一个区域内卷绕数为零，也并不意味着就不存在零点或极点<sup>1</sup>。为了改进这一算法的缺陷，我选择了先绘制可视化图像、再限定寻找零点的初始范围的方法，详见例 4.2。

总之，卷绕数应用于二维方程的求解中，具有一定的创新性和实用性。在进一步完善和优化算法的过程中，可以进一步探索其在不同领域的应用潜力。

所有代码由本人独立完成。

## 4 运行方法和参数设置

### 4.1 参量设置

关于 Pic.py:

- (1) Area: 代表要绘制的矩形区域。
- (2) p: 代表程序将会在矩形区域内均匀取  $p^2$  个采样点。

关于 Calculate.py: 只是比 Pic.py 多了一个 prec，代表精确度。

注意，如果想更改要求解的函数，直接修改 def f(z): 的定义部分即可。

### 4.2 运行方法

我们给出以下若干应用样例。

<sup>1</sup>关于极点的更多信息见附录，不在课堂上展开讲解。

**例 4.1** 求解二维方程

$$\begin{aligned} e^x + y + \sin x &= 0 \\ e^{-y} + x - \cos y &= 0 \end{aligned}$$

相当于求以下复变函数的零点：

$$f(z) = (e^x + y + \sin x) + (e^{-y} + x - \cos y)i, x = \operatorname{Re}\{z\}, y = \operatorname{Im}\{z\}$$

先在一个比较大的区域内绘制图像：设定  $\text{Area} = [[-3, -3], [3, 3]]$ ，运行 Pic.py 得到的图像为图 3，可以看出函数在这个区域上有一个零点，再运行 Calculate.py，得到的算法轨迹图为图 4。

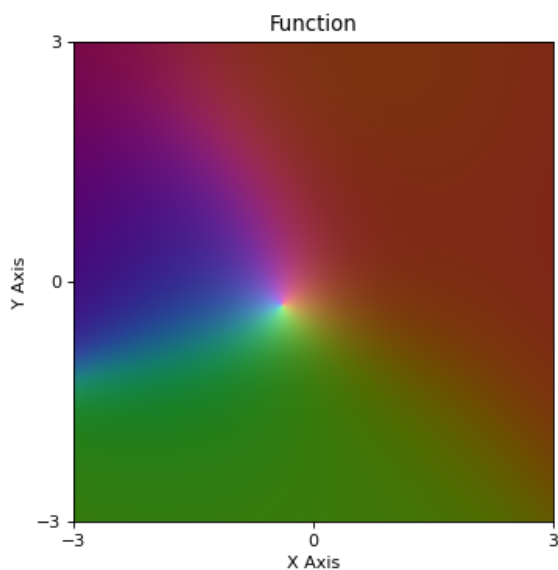
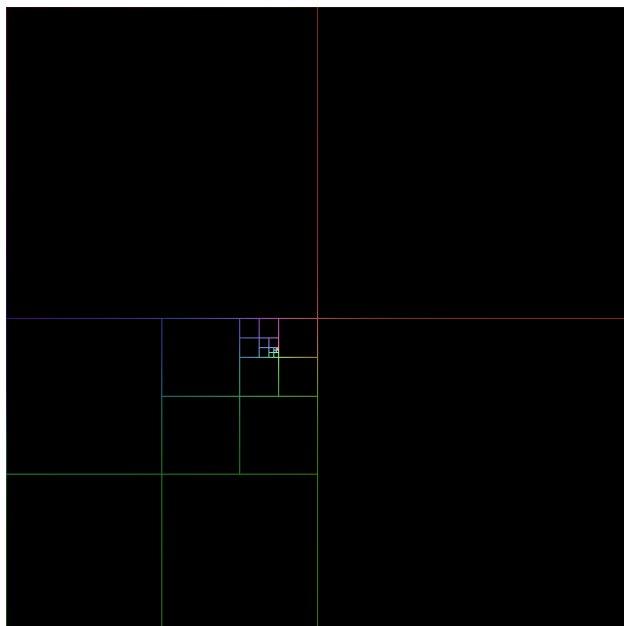
图 3:  $f(z)$  图像

图 4: 算法轨迹

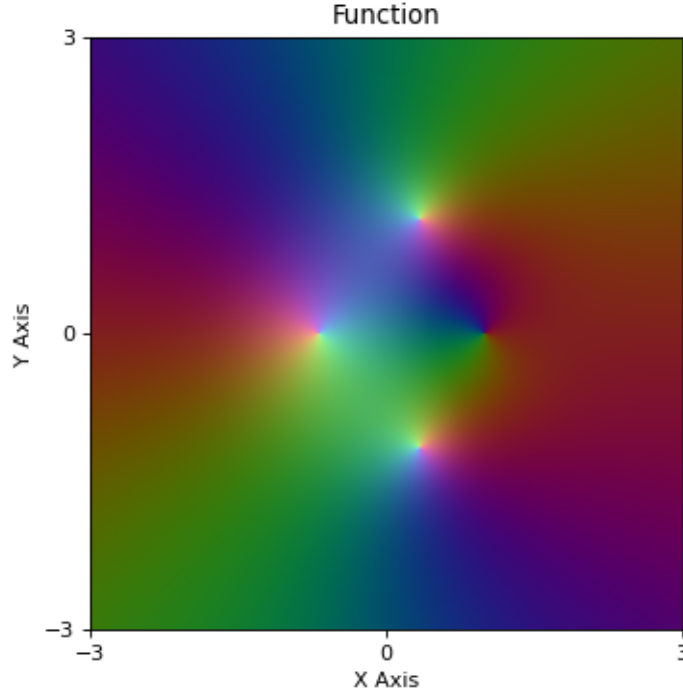
$$z = (-0.38983 - 0.29714j)$$

$$1\text{阶零点}, f(z) = (8.088608351997095e - 07 - 4.041852305736171e - 06j)$$

说明零点位于  $z = -0.38983 - 0.29714i$ ，原方程的数值解为  $x = -0.38983, y = -0.29714$ 。

**例 4.2** 求  $f(z) = z^2 + z + \frac{3}{z-1} + 2$  的所有零点。

先在一个比较大的区域内绘制图像：设定  $\text{Area} = [[-3, -3], [3, 3]]$ ，运行 Pic.py 得到的图像为

图 5:  $f(z), z \in Area$ 

图中的黑点代表极点，为了避免极点对算法的影响，我们选择新的区域来避开极点：  
设定  $Area = [[-3, -3], [0.7, 3]]$ ，再运行 Calculate.py，输出结果为：

$$z = (0.34116 - 1.16154j)$$

$$1\text{阶零点}, f(z) = (-8.702062850529657e - 06 + 6.994314995178996e - 06j)$$

$$z = (-0.68233 + 0j)$$

$$1\text{阶零点}, f(z) = (3.1287591240047874e - 06 + 0j)$$

$$z = (0.34116 + 1.16154j)$$

$$1\text{阶零点}, f(z) = (-8.702062850529657e - 06 - 6.994314995178996e - 06j)$$

说明零点位于  $z = 0.34116 \pm 1.16154i, -0.68233$ .

**例 4.3** 黎曼 Zeta 函数：

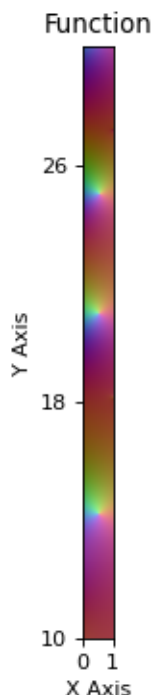
$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}, s \in \mathbb{R} \text{ and } s > 1$$

它可以全纯延拓到整个复平面上，除了点  $s = 1$ ，例如

$$\zeta(s) = \frac{1}{1 - 2^{1-s}} \sum_{n=1}^{\infty} \frac{(-1)^{n-1}}{n^s}, 0 \leq \text{Re}\{s\} \leq 1, s \neq 1$$

黎曼猜想提出  $\zeta$  的所有非平凡零点都满足实部为  $1/2$ ，我们不妨取该级数的前 20 项<sup>2</sup>，在区域  $\{x \in [0, 1], y \in [10, 30]\}$  上寻找零点，看看是否符合猜想：

<sup>2</sup>这是一个极其粗糙的近似，在这里只是为了举例。

图 6:  $f(z), z \in Area$ 

可以看出零点的确大致分布在  $1/2$  处, 运行 Calculate.py, 输出结果为:

$$z = (0.52059 + 14.19055j)$$

$$1\text{阶零点}, f(z) = (-2.8647281241464904e - 06 - 1.8098904839814752e - 06j)$$

$$z = (0.52614 + 20.9771j)$$

$$1\text{阶零点}, f(z) = (-3.3416288266136102e - 06 - 1.1850571002382669e - 06j)$$

$$z = (0.54841 + 25.0335j)$$

$$1\text{阶零点}, f(z) = (2.867998343922471e - 06 - 2.7305839407558273e - 06j)$$

算法找到的三个零点实部分别是 0.52059, 0.52614, 0.54841.

## 参考资料

《复变函数》史济怀、刘太顺

[https://www.zhihu.com/column/c\\_1419093555923615744](https://www.zhihu.com/column/c_1419093555923615744)

## 附录：关于亚纯函数与本性奇点

亚纯函数是指那些在区域内除有限个孤立奇点之外全纯的函数, 且那些孤立奇点都是极点。对于极点来说, 对本算法有着一定影响, 但总是可以通过调整初始范围来避免, 这是因为全纯函数零点的孤立性, 以及亚纯函数极点的孤立性。值得一提的是, 根据辐角原理, 如果零点和极点离得足够远, 本文中的算法也会收敛到极点。

而对于本性奇点, 算法无法消除它带来的影响。根据 Picard 大定理,  $f(z)$  在它的本性奇点的附近可以对任意复值取到无穷多次, 最多只有一个例外。这意味着本性奇点的附近可能会有无穷多个零点<sup>3</sup>, 此时算法失效。这是因为本性奇点附近的零点没有孤立性, 算法会因为精度问题将靠近的几个零点认定为同一个零点。例如下面这个例子:

<sup>3</sup>即使 0 是例外, 本性奇点附近的辐角变化速度也相较于其他区域急剧增加, 因此由于精度问题算法也会错误地算出很多零点。

例 4.4 求  $f(z) = e^{\frac{1}{z}} - z$  的零点, 区域设定为  $Area = [[-2, -2], [2, 2]]$ .

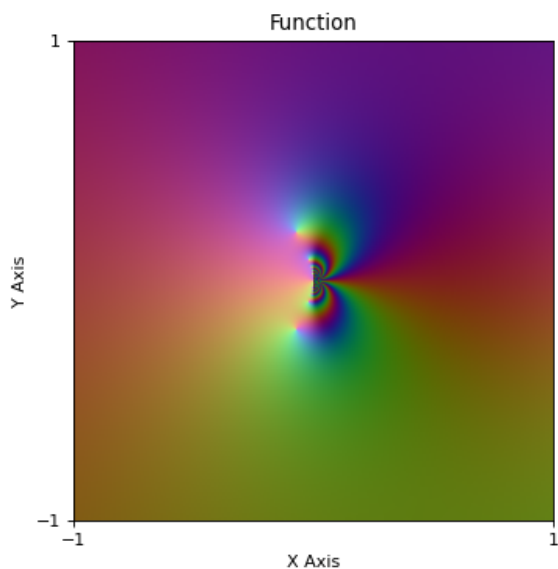


图 7:  $f(z)$  图像

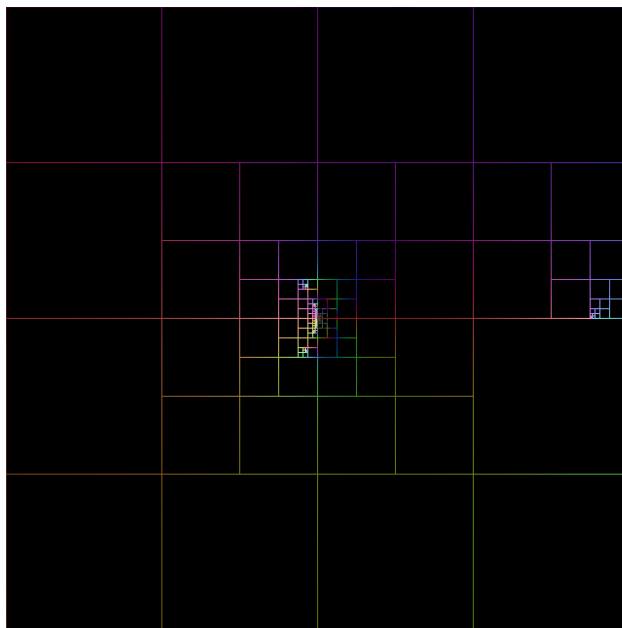


图 8: 算法轨迹

程序输出也很长, 我这里节选一部分:

```

z = (-0.07136 - 0.20354j)
1阶零点, f(z) = (1.380705354953049e - 05 - 2.5436442745441212e - 06j)
z = (-0.0197 - 0.08841j)
1阶零点, f(z) = (-4.5761381641237775e - 05 - 2.654443479493951e - 05j)
z = (-0.00948 - 0.05685j)
1阶零点, f(z) = (4.370789652767722e - 05 + 7.636534046943921e - 06j)
:
z = (-1e - 05 - 0.00123j)
10阶零点, f(z) = (-0.0010048634612231583 + 0.002116701948244485j)
函数在给定区域内可能存在本性奇点, 请更换区域重试!

```

算法发现了超过 10 阶的零点时, 就会认为区域内存在本性奇点。

时间原因, 这部分不作为课堂展示时的讲解内容。