# 程序介绍与算法说明

本程序的目的是计算对数螺旋的参数并绘制相应图形，主要流程如下：

1. 定义中心螺旋线长度函数：
   * 公式：
   * 实现：
   * # 实现：中心螺旋线长度函数实现  
     def Lc(b):  
      return np.sqrt(1 + b\*\*2) \* (s\_max - s\_min) / (2 \* b) \* (np.exp(2\*np.pi\*b) + 1) / (np.exp(2\*np.pi\*b) - 1)
2. 利用 Brent 法求解参数 b，使得 L\_c(b) 等于预设的中心螺旋线长度 L\_center。
   * 代码中使用了 scipy.optimize.brentq 求解零点。
3. 根据求得的 b，计算参数 a，其公式为：
   * 公式：
4. 计算终止角度 θ\_end：
   * 公式：
5. 将螺旋线在区间 [0, θ\_end] 内等分为若干小段，分别计算每个节点的边缘和中心径向坐标。
6. 利用极坐标到笛卡尔坐标的转换公式：
   * 公式：
7. 绘制每一小段四边形，并将图形导出为 DXF 文件，方便后续的工程应用。
8. 边缘螺旋线设计过程：
   * 公式：
   * 设计过程：利用指数函数描述螺旋线的扩张，确保在旋转过程中每一圈的径向距离逐渐增大，从而满足最小和最大间距的要求。
9. 中心螺旋线设计过程：
   * 公式：
   * 设计过程：通过取相邻两圈边缘的平均值，获得中间轨迹，实现平滑过渡及增强结构稳定性。
10. 设计过程实现代码：
    * 以下代码片段展示了如何计算边缘螺旋线和中心螺旋线：
    * # 实现：计算边缘与中心螺旋线  
      r\_edge\_seg = a \* np.exp(b \* theta\_seg)  
      r\_center\_seg = 0.5 \* (r\_edge\_seg + a \* np.exp(b \* (theta\_seg + 2\*np.pi)))

计算程序：

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.optimize import brentq  
from solid import polygon, rotate, rotate\_extrude, translate, union, scad\_render\_to\_file  
from pathlib import Path  
import ezdxf  
  
# ─────────────指定参数 ──────────────────  
s\_min = 12.0 # 最小圈间径向间距 (mm)  
s\_max = 66.0 # 最大圈间径向间距 (mm)  
L\_center = 200.0 # center 螺旋线长度 (mm)  
num\_segments = 12 # 将螺旋分成多少段，可按需修改  
# ────────────────────────────────────────  
  
# 根据指定的 center 螺旋线长度计算 b, a, θ\_end  
  
def Lc(b):  
 # 中心螺旋线长度函数 L\_c(b)  
 # LaTeX: L\_c(b) = \int\_0^{2\pi} \sqrt{r(\theta)^2 + \bigl(r'(\theta)\bigr)^2}\,d\theta  
 # 经过解析可得：  
 # L\_c(b) = \sqrt{1 + b^2}   
 # \,\frac{s\_{\max} - s\_{\min}}{2b}  
 # \,\frac{e^{2\pi b} + 1}{e^{2\pi b} - 1}  
 return np.sqrt(1 + b\*\*2) \* (s\_max - s\_min) / (2 \* b) \* (np.exp(2\*np.pi\*b) + 1) / (np.exp(2\*np.pi\*b) - 1)  
  
# 求解 b 使得 L\_c(b) = L\_center  
# LaTeX: \text{find } b: L\_c(b) - L\_{\rm center} = 0  
b = brentq(lambda bb: Lc(bb) - L\_center, 1e-6, 1.0)  
  
# 计算 a  
# LaTeX: a = \frac{s\_{\min}}{e^{2\pi b} - 1}  
a = s\_min / (np.exp(2\*np.pi\*b) - 1)  
  
# 计算 θ\_end  
# LaTeX: \theta\_{\rm end} = \frac{1}{b} \ln\!\bigl(\tfrac{s\_{\max}}{s\_{\min}}\bigr)  
theta\_end = np.log(s\_max / s\_min) / b  
  
print(f"计算结果： a = {a:.4f}, b = {b:.4f}, θ\_end = {theta\_end:.4f}")  
  
# 将螺旋线分割成等角度小段  
# LaTeX: \theta\_i = i\,\Delta\theta,\quad \Delta\theta = \frac{\theta\_{\rm end}}{\text{num\\_segments}}  
theta\_seg = np.linspace(0, theta\_end, num\_segments + 1)  
  
# 计算每个节点的径向坐标  
# LaTeX: r\_{\rm edge}(\theta) = a\,e^{b\theta}  
# LaTeX: r\_{\rm center}(\theta) = \tfrac12\bigl(r\_{\rm edge}(\theta) + r\_{\rm edge}(\theta + 2\pi)\bigr)  
r\_edge\_seg = a \* np.exp(b \* theta\_seg)  
r\_center\_seg = 0.5 \* (r\_edge\_seg + a \* np.exp(b \* (theta\_seg + 2\*np.pi)))  
  
# 极坐标转笛卡尔坐标  
# LaTeX: x(\theta) = r(\theta)\cos\theta,\quad y(\theta) = r(\theta)\sin\theta  
x\_edge\_seg = r\_edge\_seg \* np.cos(theta\_seg)  
y\_edge\_seg = r\_edge\_seg \* np.sin(theta\_seg)  
x\_ctr\_seg = r\_center\_seg \* np.cos(theta\_seg)  
y\_ctr\_seg = r\_center\_seg \* np.sin(theta\_seg)  
  
# 绘制每个小段为四边形，连接相同 θ 的 edge 和 center 点  
plt.figure(figsize=(6,6))  
for i in range(num\_segments):  
 xs = [  
 x\_edge\_seg[i],  
 x\_edge\_seg[i+1],  
 x\_ctr\_seg[i+1],  
 x\_ctr\_seg[i],  
 x\_edge\_seg[i]  
 ]  
 ys = [  
 y\_edge\_seg[i],  
 y\_edge\_seg[i+1],  
 y\_ctr\_seg[i+1],  
 y\_ctr\_seg[i],  
 y\_edge\_seg[i]  
 ]  
 plt.plot(xs, ys, '-k')  
  
plt.axis('equal')  
plt.xlabel('X (mm)')  
plt.ylabel('Y (mm)')  
plt.title(f'对数螺旋细分为{num\_segments}个四边形')  
plt.grid(True)  
  
# 新增：导出 DXF 文件  
doc = ezdxf.new(dxfversion='R2010')  
msp = doc.modelspace()  
for i in range(num\_segments):  
 pts = [  
 (x\_edge\_seg[i], y\_edge\_seg[i]),  
 (x\_edge\_seg[i+1], y\_edge\_seg[i+1]),  
 (x\_ctr\_seg[i+1], y\_ctr\_seg[i+1]),  
 (x\_ctr\_seg[i], y\_ctr\_seg[i]),  
 (x\_edge\_seg[i], y\_edge\_seg[i])  
 ]  
 msp.add\_lwpolyline(pts, close=True)  
  
dxf\_output\_path = "line.dxf"  
doc.saveas(dxf\_output\_path)  
print(f"DXF文件已导出到 {dxf\_output\_path}")  
  
plt.show()