波浪能最大输出功率设计

# 摘 要

**关键词：**

# 一、问题重述

# 二、问题分析

# 三、模型假设

# 四、符号约定

|  |  |  |
| --- | --- | --- |
| 符号 | 定义 | 单位 |
|  | 振子的质量 | Kg |
|  | 浮子的质量 | Kg |
|  | 重力加速度 |  |
|  | 液体的密度 |  |
|  | 垂荡附加质量 |  |
|  | i物体运动的加速度 |  |
|  | 兴波阻尼系数 |  |

# 五、模型的建立与求解

## 5.1问题一

### 5.1.1问题一模型的建立（取上为正方向）

### 5.1.1.1初始时刻状态确立

初始时刻波浪能装置平衡于静水中，对系统整体受力分析，仅受重力作用和浮力作用。 （受力分析图）

其中是系统所受的浮力。

根据浮力的定义式，是系统此时的排水量，即系统浸入水中的体积。

单独对振子受力分析确定弹簧的形变量：

### 5.1.1.2运动模型建立

隔离法分别浮子和振子受力分析：

对浮子：

代入有

=

对于振子：

### 5.1.1.3排水量模型

由于浮子的垂荡位移会改变系统的排水量，即是关于的函数，代码如下：

def solve\_v(x, l, h, r, h2): # 根据浮子的位移确定排水量

if x >= l:

return 0

elif l > x >= l - h:

return 1 / 3 \* math.pi \* r \* r / (h \* h) \* (l - x) \*\* 3

elif l - h2 - h <= x < l - h:

return 1 / 3 \* math.pi \* r \* r \* h + math.pi \* r \* r \* (l - x - h)

elif x < l - h2 - h:

return 1 / 3 \* math.pi \* r \* r \* h + math.pi \* r \* r \* h2

### 5.1.2问题一模型的求解

利用四阶五阶嵌套的龙格-库塔方法求解5.1.1.2中建立的运动方程。

将浮子与振子的运动方程变形；

=

10 s、20 s、40 s、60 s、100 s 时，浮子与振子的垂荡位移和速度如下表所示：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 浮子 | | 振子 | |
|  | 位移 | 速度 | 位移 | 速度 |
| t=10 | -0.19053 | -0.64002 | -0.21151 | -0.69316 |
| T=20 | -0.59041 | -0.24022 | -0.63403 | -0.27073 |
| T=40 | 0.285469 | 0.313847 | 0.296628 | 0.333698 |
| T=60 | -0.31446 | -0.47869 | -0.33131 | -0.51563 |
| T=100 | -0.0836 | -0.60392 | -0.08403 | -0.64293 |

### 5.2.1问题二模型的建立

建立以制程界限为约束条件，传送带过炉速度最大为目标的优化模型。

决策变量：传送带过炉速度v

目标函数：

约束条件：

同时应满足模型一建立的焊接板中心随时间变化的模型。

### 5.2.2问题二模型的求解

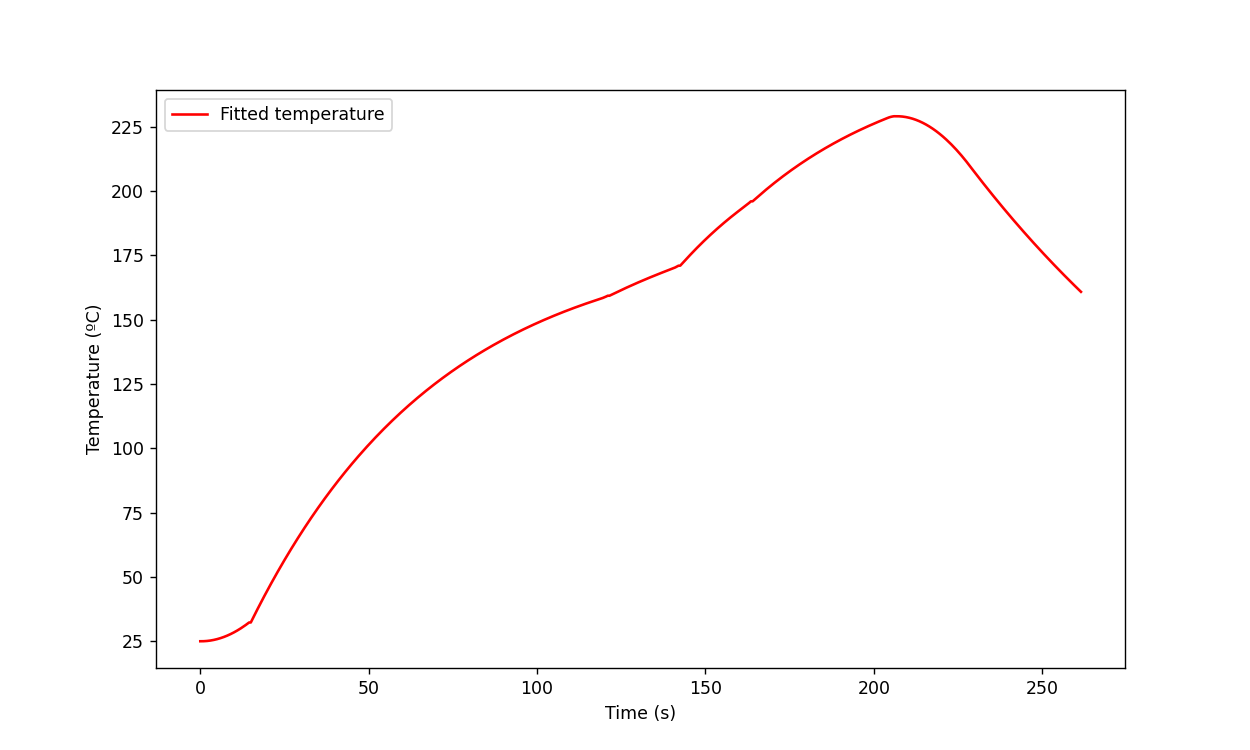
借助Python，直接迭代遍历满足约束条件的最大传送带过炉速度。解得，最大过炉速度为71.80cm/min。此时，炉温曲线如下图所示：

图3最大过炉速度时的炉温曲线

### 5.3.1问题三模型的建立

建立以各温区温度，传送带过炉速度为决策变量，以超过217ºC到峰值温度所覆盖的面积最小为目标函数，以制程界限和各温区温度，传送带过炉速度满足条件为约束的多决策变量的优化模型即：

决策变量：传送带过炉速度v，各个温区的设定温度

目标函数：

约束条件：

### 5.3.2问题三模型的求解

问题三涉及比较多的决策变量，如果继续利用Python，直接迭代遍历满足约束条件来求解的话，不能保证求解的速度和准确性。因此利用差分进化算法不断探求全局最优解，最终求得当超过217ºC到峰值温度所覆盖的面积最小时，各温区设定的温度如下表所示：

表3各温区设定的温度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 温区 | 预热区 | 恒温区 | 回流区 | 冷却区 |
| 设定温度（℃） | 183.74518715 | 203.2048051 | 244.82480274 | 264.99988482 |

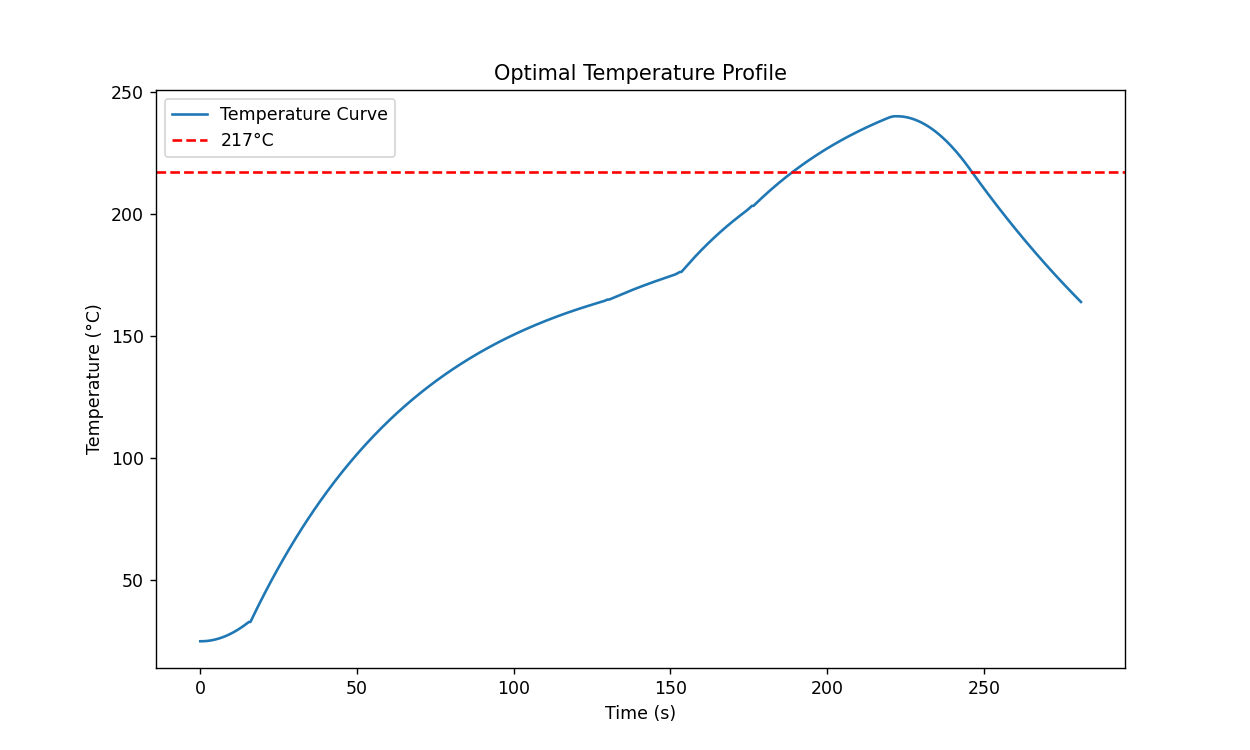
此时传送带的过炉速度为92.90 cm/min，最小阴影面积为412.16℃·S。炉温曲线如下图所示：

图4最小阴影面积下的炉温曲线

### 5.4.1问题四模型的建立

### 5.4.1.1对称差异性指标

利用峰值温度为中心线的两侧超过217ºC的炉温曲线的差值构建对称差异性指标，则**两侧差值越小，说明炉温曲线越对称。**

### 5.4.1.2考虑对称性的改进优化模型

建立以各温区温度，传送带过炉速度为决策变量，以超过217ºC到峰值温度所覆盖的面积和对称性指标最小为目标函数，以制程界限和各温区温度，传送带过炉速度满足条件为约束的多决策变量的优化模型即：

决策变量：传送带过炉速度v，各个温区的设定温度

目标函数：

约束条件：

### 5.4.2问题四模型的求解

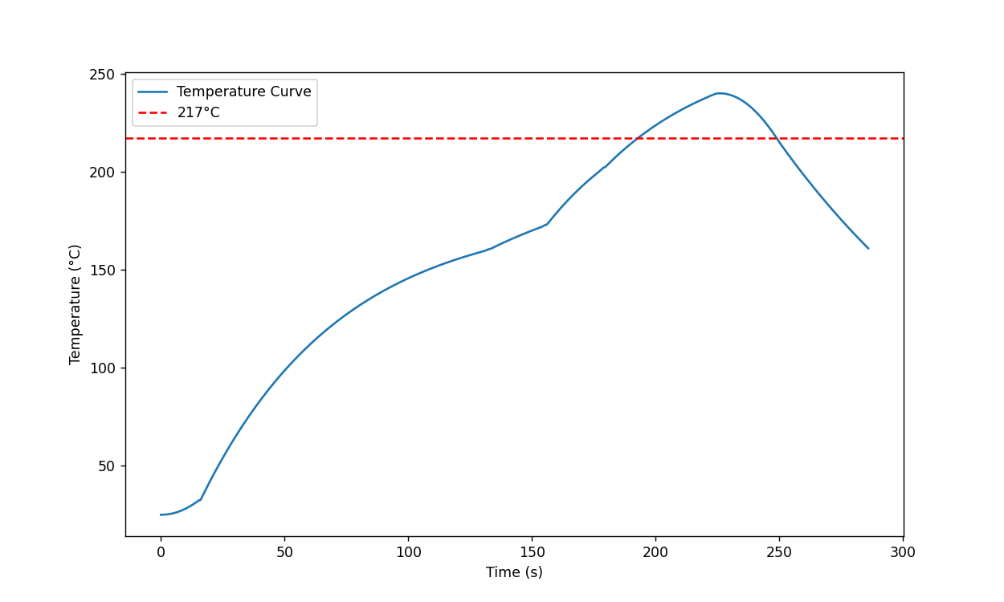
同样，利用Python中的差分进化算法全局搜索最优解[3]。

结果如下所示：

表4各温区设定的温度

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 温区 | 预热区 | 恒温区 | 回流区 | 冷却区 |
| 设定温度（℃） | 179.65711113 | 203.89987803 | 243.63604457 | 264.99999931 |

此时传送带的过炉速度为91.26 cm/min，最小阴影面积为414.59℃·S，对称差异性指标为5.23℃。炉温曲线如下图所示：



# 六、模型的评价

## 6.1模型的优点

（1）在使用牛顿冷却定律的基础上，根据问题的实际情况，分段进行拟合，使得模型总体更好地符合实际情况。

（2）使用差分方程对模型进行拟合，使得前一个分段的温度情况更好地影响到下一个分段的温度，体现了温度变化的连续性和在不同温度环境下的变化性。

（3）在原有模型基础上增加约束条件，体现了建模过程的连贯性，简化了模型建立过程。

（4）将题目中的限制条件添加到差分进化算法中，在全局中寻找限制条件下的最优解。

（5）成功将图形上的对称性用差分结果的对称差异性指标来表示。

## 6.2模型的缺点

1. 在后续问题的求解中沿用了第一问中求得的k值，忽略了后面由于温度变化而导致的k值的变化。
2. 由于对过程进行分段时，需要将分段结果离散化和对时间取整，这使得模型得到的拟合曲线在分段点出体现出离散和间断的结果。

# 七、参考文献

1. 詹士昌.牛顿冷却定律适用范围的探讨[J].大学物理,2000,(05):36-37.DOI:10.16854/j.cnki.1000-0712.2000.05.012.
2. 龚翌晖,杨祺.Fermat型复微分差分方程的整函数解[J].新疆师范大学学报(自然科学版),2024,43(03):69-74.DOI:10.14100/j.cnki.1008-9659.20240514.001.
3. 杨启文,蔡亮,薛云灿.差分进化算法综述[J].模式识别与人工智能,2008,21(04):506-513.

# 八、附录

## 问题1.py

from scipy.optimize import curve\_fit

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.metrics import mean\_squared\_error

v = 70 / 60 # cm/s 速度

temp = [25, 175, 175, 175, 175, 175, 195, 235, 255, 255, 25, 25, 25] # 温度

# 读取 Excel 文件

file\_path = '附件.xlsx'

data = pd.read\_excel(file\_path)

# 查看前几行数据

print(data.head())

# 提取时间和温度数据

time = data['时间(s)'].values

temperature = data['温度(ºC)'].values

def external\_temperature(s):

""" 根据位移 s 计算外界温度 T\_ext """

if s <= 25:

return (temp[1] - temp[0]) / 25 \* s + temp[0]

elif 25 < s <= 197.5:

return temp[1]

elif 197.5 < s <= 202.5:

return (temp[6] - temp[5]) / 5 \* (s - 197.5) + temp[5]

elif 202.5 < s <= 233:

return temp[6]

elif 233 < s <= 238:

return (temp[7] - temp[6]) / 5 \* (s - 233) + temp[6]

elif 238 < s <= 268.5:

return temp[7]

elif 268.5 < s <= 273.5:

return (temp[8] - temp[7]) / 5 \* (s - 268.5) + temp[7]

elif 273.5 < s <= 339.5:

return temp[8]

elif 339.5 < s <= 380:

return (temp[10] - temp[9]) / 40.5 \* (s - 339.5) + temp[9]

else:

return 25

def temperature\_model(t, k , T\_init):

T = np.zeros\_like(t)

T[0] = T\_init # 初始温度

dt = t[1] - t[0] # 时间步长

for i in range(1, len(t)):

s = t[i - 1] \* v

T\_ext = external\_temperature(s)

T[i] = T[i - 1] + k \* (T\_ext - T[i - 1]) \* dt

return T

# 定义区间

segments = [

(0, int(25/v-19)\*2),

(int(25/v-19)\*2, int((25+35.5\*5-5)/v-19)\*2),

( int((25+35.5\*5-5)/v-19)\*2, int((25+35.5\*6-5)/v-19)\*2),

( int((25+35.5\*6-5)/v-19)\*2, int((25+35.5\*7-5)/v-19)\*2),

( int((25+35.5\*7-5)/v-19)\*2, int((25+35.5\*9-5)/v-19)\*2),

( int((25+35.5\*9-5)/v-19)\*2,len(time))

]

# 存储分段拟合结果

all\_fitted\_temps = []

overall\_rmse = 0

for start, end in segments:

time\_segment = time[start:end]

temp\_segment = temperature[start:end]

T\_init\_segment = temperature[start] # 初始温度

# 非线性拟合

popt, pcov = curve\_fit(lambda t, k: temperature\_model(t, k, T\_init\_segment), time\_segment, temp\_segment, p0=[0.1])

K\_fit = popt[0]

print(f"Fitted K value: {K\_fit:.4f}")

fitted\_temp\_segment = temperature\_model(time\_segment, K\_fit, T\_init\_segment)

all\_fitted\_temps.extend(fitted\_temp\_segment)

# 计算该段的RMSE

segment\_rmse = np.sqrt(mean\_squared\_error(temp\_segment, fitted\_temp\_segment))

print(f"Segment RMSE: {segment\_rmse:.4f}")

overall\_rmse += segment\_rmse \*\* 2

# 计算整体的RMSE

overall\_rmse = np.sqrt(overall\_rmse / len(segments))

print(f"Overall RMSE: {overall\_rmse:.4f}")

# 将所有段的拟合结果拼接在一起

all\_fitted\_temps = np.array(all\_fitted\_temps)

plt.figure(figsize=(10, 6))

plt.plot(time, temperature, 'b-', label='original data')

plt.plot(time, all\_fitted\_temps, 'r-', label='filterd data')

plt.xlabel('Time (s)')

plt.ylabel('Temperature (ºC)')

plt.legend()

plt.show()

## 问题2.py

import numpy as np

import matplotlib.pyplot as plt

# 设定的温度

temp = [25, 182, 182, 182, 182, 182, 203, 237, 254, 254, 25, 25, 25]

# 外界温度函数

def external\_temperature(s):

if s <= 25:

return (temp[1] - temp[0]) / 25 \* s + temp[0]

elif s <= 197.5:

return temp[1]

elif s <= 202.5:

return (temp[6] - temp[5]) / 5 \* (s - 197.5) + temp[5]

elif s <= 233:

return temp[6]

elif s <= 238:

return (temp[7] - temp[6]) / 5 \* (s - 233) + temp[6]

elif s <= 268.5:

return temp[7]

elif s <= 273.5:

return (temp[8] - temp[7]) / 5 \* (s - 268.5) + temp[7]

elif s <= 339.5:

return temp[8]

elif s <= 380:

return (temp[10] - temp[9]) / 40.5 \* (s - 339.5) + temp[9]

else:

return 25

# 温度模型函数

def temperature\_model(t, v, k, T\_init):

T = np.zeros\_like(t)

T[0] = T\_init

dt = t[1] - t[0]

for i in range(1, len(t)):

s = t[i - 1] \* v

T\_ext = external\_temperature(s)

T[i] = T[i - 1] + k \* (T\_ext - T[i - 1]) \* dt

return T

# 检查是否满足约束条件的函数

def check\_constraints(time, Tem):

dt = time[1] - time[0]

rise\_time = 0

above\_217\_time = 0

peak\_temp = np.max(Tem)

for i in range(1, len(Tem)):

slope = (Tem[i] - Tem[i - 1]) / dt

if slope > 3 or slope < -3:

return False

if 150 <= Tem[i] <= 190 and slope >= 0:

rise\_time += dt

if Tem[i] > 217:

above\_217\_time += dt

if rise\_time < 60 or rise\_time > 120:

return False

if above\_217\_time < 40 or above\_217\_time > 90:

return False

if peak\_temp < 240 or peak\_temp > 250:

return False

return True

# 定义参数

l = 25 \* 2 + 35.5 \* 11 - 5

# 迭代寻找最大速度

max\_v = 0

for v in np.arange(65, 100, 0.1):

v = v / 60 # cm/s

time = np.arange(0, l / v, 0.5)

T\_init = 25

segments = [

(0, int(2 \* 25 / v), 0.0071),

(int(2 \* 25 / v), int(2 \* (25 + 35.5 \* 5) / v), 0.0176),

(int(2 \* (25 + 35.5 \* 5) / v), int(2 \* (25 + 35.5 \* 6) / v), 0.0147),

(int(2 \* (25 + 35.5 \* 6) / v), int(2 \* (25 + 35.5 \* 7) / v), 0.0223),

(int(2 \* (25 + 35.5 \* 7) / v), int(2 \* (25 + 35.5 \* 9) / v), 0.0203),

(int(2 \* (25 + 35.5 \* 9) / v), len(time), 0.0093),

]

Tem = np.zeros\_like(time)

for start, end, k in segments:

t\_segment = time[start:end]

T\_segment = temperature\_model(t\_segment, v, k, T\_init)

Tem[start:end] = T\_segment

T\_init = T\_segment[-1]

if check\_constraints(time, Tem):

max\_v = v \* 60

else:

continue

print(f"最大允许传送带速度: {max\_v:.2f} cm/s")

plt.figure(figsize=(10, 6))

plt.plot(time, Tem, 'r-', label='Fitted temperature')

plt.xlabel('Time (s)')

plt.ylabel('Temperature (ºC)')

plt.legend()

plt.show()

## 问题3.py

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import differential\_evolution

from scipy.integrate import simpson

# 设定的温度

temp = np.zeros(12)

# 定义参数

l = 25 \* 2 + 35.5 \* 11 - 5

# 外界温度函数

def external\_temperature(s):

if s <= 25:

return (temp[1] - temp[0]) / 25 \* s + temp[0]

elif s <= 197.5:

return temp[1]

elif s <= 202.5:

return (temp[6] - temp[5]) / 5 \* (s - 197.5) + temp[5]

elif s <= 233:

return temp[6]

elif s <= 238:

return (temp[7] - temp[6]) / 5 \* (s - 233) + temp[6]

elif s <= 268.5:

return temp[7]

elif s <= 273.5:

return (temp[8] - temp[7]) / 5 \* (s - 268.5) + temp[7]

elif s <= 339.5:

return temp[8]

elif s <= 380:

return (temp[10] - temp[9]) / 40.5 \* (s - 339.5) + temp[9]

else:

return 25

# 温度模型函数

def temperature\_model(t, v, k, T\_init):

T = np.zeros\_like(t)

T[0] = T\_init

dt = t[1] - t[0]

for i in range(1, len(t)):

s = t[i - 1] \* v

T\_ext = external\_temperature(s)

T[i] = T[i - 1] + k \* (T\_ext - T[i - 1]) \* dt

return T

def check\_constraints(time, Tem):

dt = time[1] - time[0]

rise\_time = 0

above\_217\_time = 0

peak\_temp = np.max(Tem)

for i in range(1, len(Tem)):

slope = (Tem[i] - Tem[i - 1]) / dt

if slope > 3 or slope < -3:

return False

if 150 <= Tem[i] <= 190 and slope >= 0:

rise\_time += dt

if Tem[i] > 217:

above\_217\_time += dt

if rise\_time < 60 or rise\_time > 120:

return False

if above\_217\_time < 40 or above\_217\_time > 90:

return False

if peak\_temp < 240 or peak\_temp > 250:

return False

return True

# 目标函数

def objective(params):

temp\_settings, speed = params[:-1], params[-1]

# 调整设定温度

temp[0] = 25

temp[1:6] = temp\_settings[0]

temp[6] = temp\_settings[1]

temp[7] = temp\_settings[2]

temp[8:10] = temp\_settings[3]

temp[10:13] = 25

v = speed / 60 # cm/s

time = np.arange(0, l / v, 0.5)

T\_init = 25

segments = [

(0, int(2 \* 25 / v), 0.0071),

(int(2 \* 25 / v), int(2 \* (25 + 35.5 \* 5) / v), 0.0176),

(int(2 \* (25 + 35.5 \* 5) / v), int(2 \* (25 + 35.5 \* 6) / v), 0.0147),

(int(2 \* (25 + 35.5 \* 6) / v), int(2 \* (25 + 35.5 \* 7) / v), 0.0223),

(int(2 \* (25 + 35.5 \* 7) / v), int(2 \* (25 + 35.5 \* 9) / v), 0.0203),

(int(2 \* (25 + 35.5 \* 9) / v), len(time), 0.0093),

]

Tem = np.zeros\_like(time)

for start, end, k in segments:

t\_segment = time[start:end]

T\_segment = temperature\_model(t\_segment, v, k, T\_init)

Tem[start:end] = T\_segment

T\_init = T\_segment[-1]

# 计算面积

# 找到最高温度的位置

max\_index = np.argmax(Tem)

# 只考虑最大值左边的部分

Tem\_left = Tem[:max\_index]

mask = Tem\_left > 217

a = Tem\_left[mask]

# 计算面积时确保 dx 是合适的

if len(a) > 0:

area = np.trapz(a, dx=0.5) - 217 \* ((len(a)-1) \* 0.5)

else:

area = 0

# 检查约束

if not check\_constraints(time, Tem):

return 1000

return area

# 定义初始值和边界

bounds = [(165, 185), (185, 205), (225, 245), (245, 265), (65, 100)]

# 使用差分进化算法进行优化

result = differential\_evolution(objective, bounds, strategy='best1bin', maxiter=10000, tol=1e-4)

# 获取最优温度设定和速度

optimal\_params = result.x

optimal\_temp\_settings = optimal\_params[:-1]

optimal\_speed = optimal\_params[-1]

print(f"最优温区设定温度：{optimal\_temp\_settings}")

print(f"最优传送带速度：{optimal\_speed:.2f} cm/min")

print(f"最优面积：{objective(optimal\_params):.2f} ")

# 绘制最优温度曲线

temp[0] = 25

temp[1:6] = optimal\_temp\_settings[0]

temp[6] = optimal\_temp\_settings[1]

temp[7] = optimal\_temp\_settings[2]

temp[8:10] = optimal\_temp\_settings[3]

temp[10:13] = 25

v = optimal\_speed / 60 # cm/s

time = np.arange(0, l / v, 0.5)

T\_init = 25

segments = [

(0, int(2 \* 25 / v), 0.0071),

(int(2 \* 25 / v), int(2 \* (25 + 35.5 \* 5) / v), 0.0176),

(int(2 \* (25 + 35.5 \* 5) / v), int(2 \* (25 + 35.5 \* 6) / v), 0.0147),

(int(2 \* (25 + 35.5 \* 6) / v), int(2 \* (25 + 35.5 \* 7) / v), 0.0223),

(int(2 \* (25 + 35.5 \* 7) / v), int(2 \* (25 + 35.5 \* 9) / v), 0.0203),

(int(2 \* (25 + 35.5 \* 9) / v), len(time), 0.0093),

]

Tem = np.zeros\_like(time)

for start, end, k in segments:

t\_segment = time[start:end]

T\_segment = temperature\_model(t\_segment, v, k, T\_init)

Tem[start:end] = T\_segment

T\_init = T\_segment[-1]

plt.figure(figsize=(10, 6))

plt.plot(time, Tem, label='Temperature Curve')

plt.xlabel('Time (s)')

plt.ylabel('Temperature (°C)')

plt.title('Optimal Temperature Profile')

plt.axhline(217, color='r', linestyle='--', label='217°C')

plt.legend()

plt.show()

## 问题4.py

import numpy as np

import matplotlib.pyplot as plt

from scipy.optimize import differential\_evolution

from scipy.integrate import simpson

# 设定的温度

temp = np.zeros(12)

# 定义参数

l = 25 \* 2 + 35.5 \* 11 - 5

# 外界温度函数

def external\_temperature(s):

if s <= 25:

return (temp[1] - temp[0]) / 25 \* s + temp[0]

elif s <= 197.5:

return temp[1]

elif s <= 202.5:

return (temp[6] - temp[5]) / 5 \* (s - 197.5) + temp[5]

elif s <= 233:

return temp[6]

elif s <= 238:

return (temp[7] - temp[6]) / 5 \* (s - 233) + temp[6]

elif s <= 268.5:

return temp[7]

elif s <= 273.5:

return (temp[8] - temp[7]) / 5 \* (s - 268.5) + temp[7]

elif s <= 339.5:

return temp[8]

elif s <= 380:

return (temp[10] - temp[9]) / 40.5 \* (s - 339.5) + temp[9]

else:

return 25

# 温度模型函数

def temperature\_model(t, v, k, T\_init):

T = np.zeros\_like(t)

T[0] = T\_init

dt = t[1] - t[0]

for i in range(1, len(t)):

s = t[i - 1] \* v

T\_ext = external\_temperature(s)

T[i] = T[i - 1] + k \* (T\_ext - T[i - 1]) \* dt

return T

def check\_constraints(time, Tem):

dt = time[1] - time[0]

rise\_time = 0

above\_217\_time = 0

peak\_temp = np.max(Tem)

for i in range(1, len(Tem)):

slope = (Tem[i] - Tem[i - 1]) / dt

if slope > 3 or slope < -3:

return False

if 150 <= Tem[i] <= 190 and slope >= 0:

rise\_time += dt

if Tem[i] > 217:

above\_217\_time += dt

if rise\_time < 60 or rise\_time > 120:

return False

if above\_217\_time < 40 or above\_217\_time > 90:

return False

if peak\_temp < 240 or peak\_temp > 250:

return False

return True

def calculate\_symmetry\_index(Tem):

mask = Tem > 217

Tem = Tem[mask]

max\_index = np.argmax(Tem)

left\_side = Tem[:max\_index + 1]

right\_side = Tem[max\_index:]

if len(left\_side) > len(right\_side):

left\_side = left\_side[:len(right\_side)]

else:

right\_side = right\_side[:len(left\_side)]

symmetry\_index = np.mean(np.abs(left\_side - right\_side[::-1]))

return symmetry\_index

def calculate\_area(Tem):

max\_index = np.argmax(Tem)

Tem\_left = Tem[:max\_index]

mask = Tem\_left > 217

a = Tem\_left[mask]

if len(a) > 0:

area = np.trapz(a, dx=0.5) - 217 \* ((len(a)-1) \* 0.5)

else:

area = 0

return area

# 目标函数

def objective(params):

temp\_settings, speed = params[:-1], params[-1]

# 调整设定温度

temp[0] = 25

temp[1:6] = temp\_settings[0]

temp[6] = temp\_settings[1]

temp[7] = temp\_settings[2]

temp[8:10] = temp\_settings[3]

temp[11:13] = 25

v = speed / 60 # cm/s

time = np.arange(0, l / v, 0.5)

T\_init = 25

segments = [

(0, int(2 \* 25 / v), 0.0071),

(int(2 \* 25 / v), int(2 \* (25 + 35.5 \* 5) / v), 0.0176),

(int(2 \* (25 + 35.5 \* 5) / v), int(2 \* (25 + 35.5 \* 6) / v), 0.0147),

(int(2 \* (25 + 35.5 \* 6) / v), int(2 \* (25 + 35.5 \* 7) / v), 0.0223),

(int(2 \* (25 + 35.5 \* 7) / v), int(2 \* (25 + 35.5 \* 9) / v), 0.0203),

(int(2 \* (25 + 35.5 \* 9) / v), len(time), 0.0093),

]

Tem = np.zeros\_like(time)

for start, end, k in segments:

t\_segment = time[start:end]

T\_segment = temperature\_model(t\_segment, v, k, T\_init)

Tem[start:end] = T\_segment

T\_init = T\_segment[-1]

# 计算面积和对称性指标

area = calculate\_area(Tem)

symmetry\_index = calculate\_symmetry\_index(Tem)

# 检查约束

if not check\_constraints(time, Tem):

return 1000

# 结合面积和对称性指标作为目标函数

return area + symmetry\_index

# 定义初始值和边界

bounds = [(165, 185), (185, 205), (225, 245), (245, 265), (65, 100)]

# 使用差分进化算法进行优化

result = differential\_evolution(objective, bounds, strategy='best1bin', maxiter=1000, tol=1e-6)

# 获取最优温度设定和速度

optimal\_params = result.x

optimal\_temp\_settings = optimal\_params[:-1]

optimal\_speed = optimal\_params[-1]

# 绘制最优温度曲线

temp[0] = 25

temp[1:6] = optimal\_temp\_settings[0]

temp[6] = optimal\_temp\_settings[1]

temp[7] = optimal\_temp\_settings[2]

temp[8:10] = optimal\_temp\_settings[3]

temp[11:13] = 25

v = optimal\_speed / 60 # cm/s

time = np.arange(0, l / v, 0.5)

T\_init = 25

segments = [

(0, int(2 \* 25 / v), 0.0071),

(int(2 \* 25 / v), int(2 \* (25 + 35.5 \* 5) / v), 0.0176),

(int(2 \* (25 + 35.5 \* 5) / v), int(2 \* (25 + 35.5 \* 6) / v), 0.0147),

(int(2 \* (25 + 35.5 \* 6) / v), int(2 \* (25 + 35.5 \* 7) / v), 0.0223),

(int(2 \* (25 + 35.5 \* 7) / v), int(2 \* (25 + 35.5 \* 9) / v), 0.0203),

(int(2 \* (25 + 35.5 \* 9) / v), len(time), 0.0093),

]

Tem = np.zeros\_like(time)

for start, end, k in segments:

t\_segment = time[start:end]

T\_segment = temperature\_model(t\_segment, v, k, T\_init)

Tem[start:end] = T\_segment

T\_init = T\_segment[-1]

# 计算最优情况下的面积和对称性指标

area = calculate\_area(Tem)

symmetry\_index = calculate\_symmetry\_index(Tem)

# 打印结果

print(f"最优温区设定温度：{optimal\_temp\_settings}")

print(f"最优传送带速度：{optimal\_speed:.2f} cm/min")

print(f"最优面积：{area:.2f}")

print(f"对称性指标：{symmetry\_index:.2f}")

plt.figure(figsize=(10, 6))

plt.plot(time, Tem, label='Temperature Curve')

plt.xlabel('Time (s)')

plt.ylabel('Temperature (°C)')

plt.axhline(217, color='r', linestyle='--', label='217°C')

plt.legend()

plt.show()