重庆科技学院

课程设计（论文）

题 目 基于遗传算法的 PID 控制器参数自整定法

院 （系） 数理与大数据学院

专业班级 大数据 2021-01

学生姓名 孟博文 学号 2021443754

2024 年 6 月 20 日

摘 要

PID 控制器作为最常用的控制器之一，在工业控制系统中有着广泛的应用。其优点是简单易实现，但其性能高度依赖于参数的选择。传统方法中，PID 参数的调节通常依赖于经验或试错法，这种方法不仅费时费力，而且对于复杂的非线性系统往往效果不佳。

为了克服传统调参方法的局限性，近年来，越来越多的研究开始探索智能优化算法在 PID 控制器参数优化中的应用。遗传算法作为一种生物启发式算法，在优化问题中展现出了良好的应用潜力。其模拟了生物进化中的遗传机制，通过不断迭代和交叉变异的过程，能够寻找到问题的较优解。

遗传算法通过建立适应度函数来评估个体的适应性，进而指导下一代个体的生成和选择。在 PID 控制器参数自整定中，适应度函数可以根据控制系统的响应特性，如稳定性、超调量和响应时间等，来评价参数组的优劣。通过不断优化适应度函数，遗传算法能够有效地搜索到较优的 PID 参数组合，从而提高控制系统的性能和稳定性。

此外，遗传算法具有并行处理能力强、全局寻优能力强等优点，使其在复杂系统和非线性系统中表现出了比传统方法更为出色的性能。因此，将遗传算法引入 PID 控制器参数自整定领域，不仅能够提高调参效率，还能够提升控制系统的鲁棒性和适应性，适用于各种工业应用和控制场景。

本文旨在探讨基于遗传算法的 PID 控制器参数自整定方法，通过对遗传算法原理和应用进行深入分析和研究，旨在为工业控制系统的优化提供新的思路和方法。通过对不同算例的实验验证和对比分析，验证遗传算法在 PID 控制器参数自整定中的有效性和实用性，为工程实践中 PID 控制器参数调节提供理论支持和实际指导。

关键词：遗传算法 PID 控制器 参数自整定

Abstract

As one of the most commonly used controllers, the PID controller is widely applied in industrial control systems. Its advantages include simplicity and ease of implementation, but its performance is highly dependent on parameter selection. In traditional methods, PID parameter tuning typically relies on experience or trial and error. This approach is not only time-consuming and labor-intensive, but it also often performs poorly for complex nonlinear systems.

To overcome the limitations of traditional parameter tuning methods, in recent years, more and more research has started to explore the application of intelligent optimization algorithms in PID controller parameter optimization. Genetic algorithms, as a type of bio-inspired algorithm, have shown great potential in optimization problems. They simulate the genetic mechanisms in biological evolution and can find optimal solutions to problems through continuous iterations and crossover mutations.

Genetic algorithms evaluate the fitness of individuals by establishing a fitness function, which in turn guides the generation and selection of the next generation of individuals. In PID controller parameter autotuning, the fitness function can be based on the response characteristics of the control system, such as stability, overshoot, and response time, to assess the quality of the parameter sets. By continuously optimizing the fitness function, genetic algorithms can effectively search for optimal PID parameter combinations, thereby improving the performance and stability of the control system.

Additionally, genetic algorithms possess strong parallel processing capabilities and robust global optimization abilities, making them perform better than traditional methods in complex and nonlinear systems. Therefore, introducing genetic algorithms into the field of PID controller parameter autotuning not only enhances tuning efficiency but also improves the robustness and adaptability of control systems. This makes them suitable for a variety of industrial applications and control scenarios.

This article aims to explore the method of PID controller parameter autotuning based on genetic algorithms. By conducting in-depth analysis and research on the principles and applications of genetic algorithms, it seeks to provide new ideas and methods for optimizing industrial control systems. Through experimental validation and comparative analysis of different cases, the effectiveness and practicality of genetic algorithms in PID controller parameter autotuning are verified. This research aims to provide theoretical support and practical guidance for PID controller parameter adjustment in engineering practice.

Keywords: Genetic Algorithm; PID Controller; Parameter Autotuning

目 录

1 研究背景与现状

1.1 PID 控制器应用背景

PID（Proportional-Integral-Derivative）控制器作为经典的控制算法，在工业控制系统中得到了广泛应用[1]，其简单性和稳定性使其成为控制工程中的重要工具。然而，PID 控制器的性能高度依赖于其参数的选择，包括比例增益、积分时间和微分时间，这些参数的不恰当选择可能导致系统响应不稳定、超调量大或者响应时间过长，从而影响控制系统的性能和效果。

1.2 PID 参数整定现状

传统的 PID 参数调节方法通常依赖于经验公式或者试错法，这种方法存在以下几个显著的缺点：一是调节过程耗时费力，需要大量的试验和实验[2]；二是难以适应复杂的非线性系统或者时变系统；三是调节结果依赖于操作人员的经验水平，缺乏系统化和科学化的方法。

2 研究目的与意义

为了克服传统调参方法的局限性，智能优化算法在 PID 控制器参数自整定中得到了广泛的关注和应用。遗传算法作为一种典型的生物启发式算法，模拟了生物进化过程中的自然选择和遗传机制。它通过不断的种群演化、交叉和变异操作，寻找最优的参数组合，从而实现对复杂问题的高效求解。

在 PID 控制器参数自整定中，遗传算法的应用主要体现在如何有效地调节 PID 控制器的参数，使得控制系统能够在面对不同的工作负载和环境变化时，保持良好的控制性能。通过定义适当的适应度函数，评估每一组参数的优劣，遗传算法能够在多维度的参数空间中搜索到最优解，同时具备较好的全局寻优能力和鲁棒性，适用于各类复杂的工业控制系统。

此外，随着计算能力的提升和算法优化的不断深入，基于遗传算法的 PID 控制器参数自整定方法在工业应用中展现出了显著的优势和潜力。它不仅能够提高调参效率，降低调试成本，还能够增强控制系统的稳定性、鲁棒性和适应性，适用于多种复杂的控制场景和工程应用中。

因此，本文旨在深入探讨基于遗传算法的 PID 控制器参数自整定方法，通过理论分析和实验验证，探索其在工业控制系统优化中的实际应用效果，为智能控制领域的研究和工程实践提供理论支持和技术指导。

3 算法介绍

3.1 PID 控制算法介绍

PID（Proportional-Integral-Derivative）控制算法是一种经典且广泛应用于工业控制系统中的反馈控制算法。它基于对系统当前状态的测量与期望状态的比较，通过调整输出控制信号来实现系统稳定性和性能的优化。典型的 PID 控制器的数学形式可以用式 3.1 表示。

$$u(t) = K_p\left(e(t) + \frac{1}{T_t}\int_0^t e(t)dt + T_D\frac{de(t)}{dt}\right) \qquad \text{公式 3.1}$$

式中包含三项，其分别为：

比例（Proportional）项：比例控制器根据当前误差的大小直接调整输出。其作用是使控制器的响应速度与误差大小成正比，这样可以快速响应系统变化，但可能会导致超调和震荡。

积分（Integral）项：积分控制器累积误差的历史值，用于消除系统稳态误差。它可以弥补由于比例控制器无法完全消除的静态误差，使系统在稳态下更为精确。

微分（Derivative）项：微分控制器根据当前误差变化的速度调整输出。它对误差的瞬时变化率进行响应，能够抑制系统的振荡和提高响应速度，但对测量噪声敏感。

PID 控制算法通过合理调节这三个参数（比例增益 Kp、积分时间 Ti、微分时间 Td），即可满足不同系统的控制要求。参数的选择通常依赖于系统的动态特性和稳定性需求，需要通过经验或者系统辨识方法来确定。

3.2 遗传算法介绍

遗传算法（Genetic Algorithm，GA）是一种受生物进化理论启发而发展起来的优化算法，用于解决复杂的搜索和优化问题。它源于达尔文的进化论思想，模拟了生物进化过程中的自然选择、遗传机制和适应度提升的过程。遗传算法通过模拟自然选择和遗传机制来寻找优化问题的最优解。其基本工作原理包括：

种群初始化：首先随机生成一个初始种群，每个个体（即解）由一组参数（称为基因）表示。

适应度评估：定义一个适应度函数来评估每个个体的优劣，通常是优化问题的目标函数或者与之相关的性能指标。

选择：根据每个个体的适应度，选择适应度较高的个体作为下一代种群的父代，采用轮盘赌或者竞争选择等策略进行选择操作。

交叉：通过交叉操作，将选定的父代个体的基因进行配对交换，生成新的后代个体。

变异：在新生成的后代个体中，以一定的概率随机改变其基因，引入新的多样性，以防止算法陷入局部最优解。

重复迭代：重复进行选择、交叉和变异操作，直到达到停止条件（如达到最大迭代次数或者目标适应度）为止。

4 构建遗传算法-PID 参数自整定模型

4.1 种群初始化

种群初始化 PID 控制器具有三个可变参数 Kp、Ki、Kd，因此初始种群中应包含三个基因，因此种群中的染色体可用图 4.1 表示

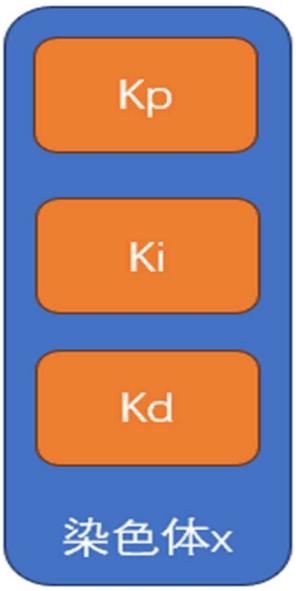

图 4.1：染色体上的基因

4.2 适应度计算

适应度函数是遗传算法的核心部分[3]，它用来评估每个个体（即一组 PID 参数）的优劣。对于 PID 控制器参数优化，适应度函数需要衡量 PID 控制器在给定系统上的性能。本项目中适应度计算主要分为以下几个部分：

4.2.1 误差计算

误差计算是指 PID 控制系统中输出值与设定值之间的差异，在本项目中采用误差平方和来评估系统的整体性能，误差平方和越小表明系统的响应越接近目标值。

在t时刻的误差e_t可用式 4.2.1.1 表示

$$e_t = target - output_t \qquad \text{公式 4.2.1.1}$$

式中：

$target$：为系统设定的目标值

$output_t$：t时刻系统的输出值

那么误差平方和可用式 4.2.1.2 表示

$$SSE = \sum_{t=0}^{T} e_t^2 \qquad \text{公式 4.2.1.2}$$

式中：

SSE：系统的误差平方和

T：系统运行时间

e_t：系统在t时刻的误差

4.2.2 过冲惩罚

过冲是指系统输出超过设定值的部分。为了避免过冲，我们需要在适应度函数中增加一个惩罚项，当系统温度超过设定值时，该惩罚项会增加。过冲惩罚可用式 4.2.2 表示。

$$Penalty_{overshoot} = \sum_{t=0}^{T} max(0, output_t - target) \qquad \text{公式 4.2.2}$$

式中：

$target$：为系统设定的目标值

$output_t$：t时刻系统的输出值

4.2.3 参数惩罚

为了避免 PID 参数过大或过小，我们增加一个基于参数大小的惩罚项。这样可以防止参数过大导致系统不稳定，或者参数过小导致系统响应太慢。参数惩罚可用式 4.2.3 表示。

$$Penalty_{param} = \alpha \cdot \left(K_p^2 + K_i^2 + K_d^2\right) \quad \text{公式 4.2.3}$$

式中：

α：参数惩罚系数

K_p：PID 积分系数

K_i：PID 微分系数

K_d：PID 比例系数

综上所述，本项目中的适应度函数可用式 4.3 表示

$$Fitness = \left(\sum_{t=0}^{T} e_t^2 + \sum_{t=0}^{T} max(0, output_t - target) + \alpha \cdot \left(K_p^2 + K_i^2 + K_d^2\right)\right)$$

公式 4.2.4

4.3 染色体选择

染色体选择用于在种群中随机选取染色体，用于之后步骤的基因交叉和变异，轮盘赌选择（Roulette Wheel Selection）是遗传算法中的一种常用选择方法，它根据适应度值的大小来选择个体，适应度值越高，被选择的概率就越大，其基本流程如下：

计算种群的总适应度，其公式表示为

$$F = f_1 + f_2 + \cdots + f_n \quad \text{公式 4.3.1}$$

式中：

f：种群中每个染色体的适应度

计算选择概率，其公式表示为

$$p_i = \frac{f_i}{F} \quad \text{公式 4.3.2}$$

式中：

f_i：第个i个体的适应度

计算累积概率，其公式表示为

$$C_N = p_1 + p_2 + \cdots + p_n \quad \text{公式 4.3.3}$$

式中：

p：第n个个体被选中的概率

此时我们生成一个随机数r满足如下条件：

$$r \in [0,1) \quad \text{公式 4.3.4}$$

这时被选中的个体为满足$C_i \geq r$的第一个染色体i

4.4 基因交叉：

基因交叉是指将两个个体的基因片段在某一点或者某几点进行互换，在本项目中采用单点交叉的方式交叉两个染色体上的基因，基因交叉的示意图如图 4.4 所示。

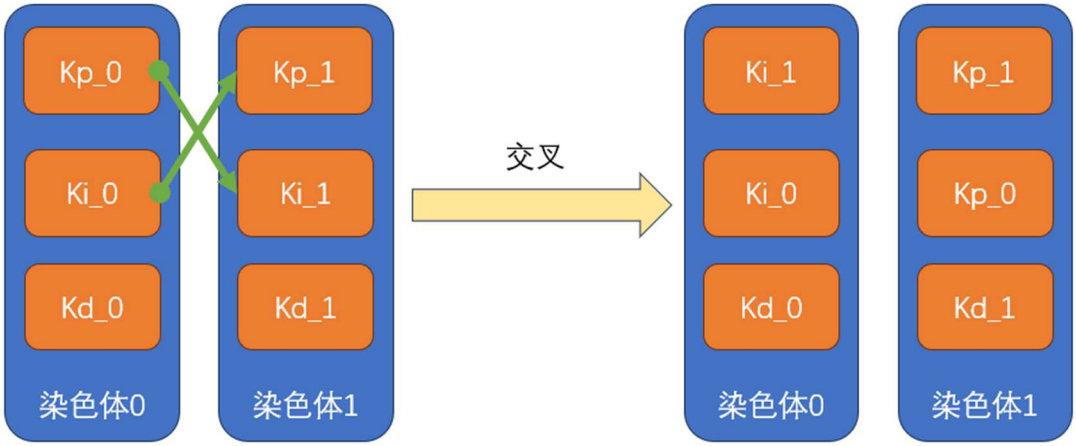

图 4.4：基因交叉示意图

从图中可见，基因交叉步骤大大增加了种群中染色体的基因序列的丰富程度，式优化的重要手段。

4.5 基因变异

变异在遗传过程中属于小概率事件，但是在种群数量较小的情况下，只通过交叉操作并不能产生优秀的后代，此时变异就显得非常重要了。通过适当的变异甚至能够产生更优秀的后代。变异的方式有很多种，常规的变异有基本位变异和逆转变异。在本项目中采用基本位变异，其示意图如图 4.5 所示

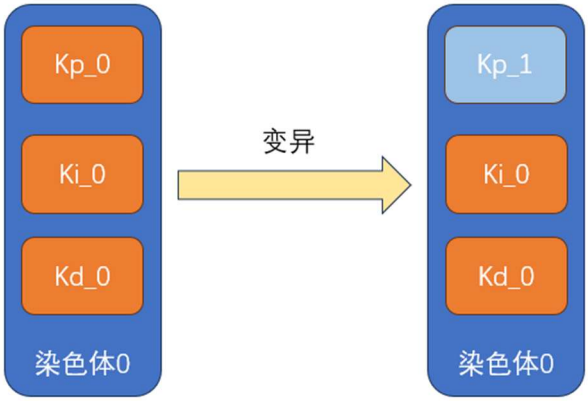

图 4.5：基因基本位变异

综上所述，使用遗传算法进行 PID 控制器参数自整定的流程如图 4.6 所示：

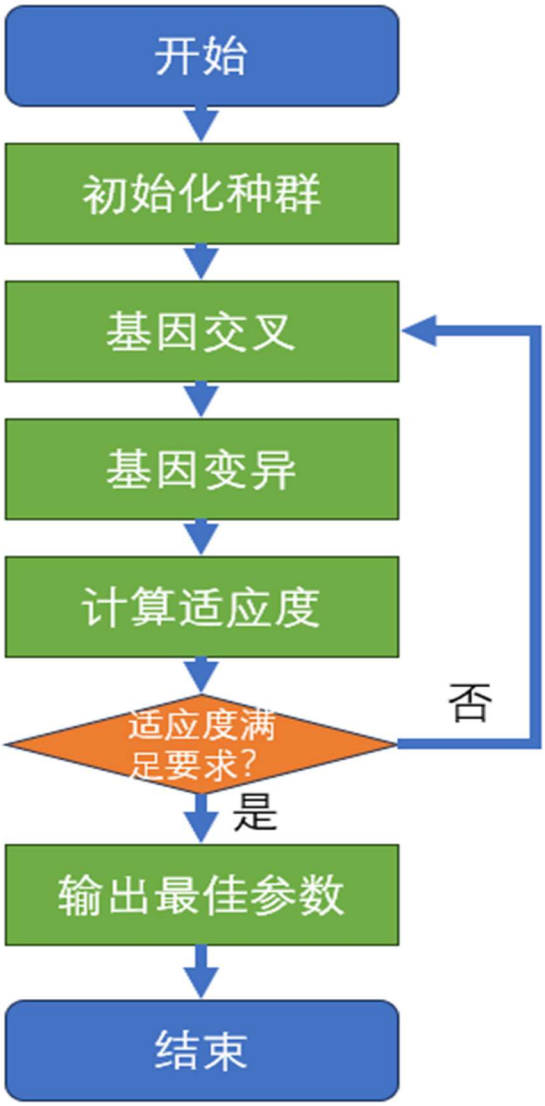

图 4.6：遗传算法进行 PID 参数自整定流程图

5 模型仿真

5.1 构建 PID 温度控制系统

PID 温度控制系统中包含一个加热器和温度传感器[4]。加热器用于接收 PID 控制器传出的加热信号对物体进行加热，温度传感器用于测量物体的实时温度并作为反馈传入 PID 控制器中。在t时刻，该温度控制系统的温度T_t满足公式 5.1.1：

$$T_t = T_{t-1} + \frac{P \times p}{C \times T} \qquad \text{公式 5.1.1}$$

式中：

T_{t-1}为$t-1$时刻时系统的温度

P为系统的输入功率

p为系统的加热功率

C为被加热物体的热容

T为采样时间间隔

该加热系统采用 PID 控制器为连续理想型 PID 控制器，用公式 5.1.2 表示：

$$u(t) = K_p\left(e(t) + \frac{1}{T_t}\int_0^t e(t)dt + T_D\frac{de(t)}{dt}\right) \qquad \text{公式 5.1.2}$$

式中：

K_p：比例增益，与比例度成倒数关系

T_t：积分时间常数

T_D：微分时间常数

$u(t)$：PID 控制器的输出

$e(t)$：给定值与测量值的误差

将连续状态的公式进行整理可得

$$u(x) = K_p\left(err(t) + \frac{1}{T}\int err(t)dt + \frac{T_d err(t)}{dt}\right) \qquad \text{公式 5.1.3}$$

当采集数据的时间间隔为T，则在第k_T时刻我们不妨提出如下假设：

误差等于第k个周期时刻的误差等于输入值减去输出值，即：

$$err(k) = rin(k) - rout(k) \qquad \text{公式 5.1.4}$$

积分环节为所有时刻的误差之和，即：

$$err(k) = err(k+1) + err(k+2) + \cdots \qquad \text{公式 5.1.5}$$

微分环节为第k时刻的误差变化率，即：

$$\frac{err(k)-err(k-1)}{T}$$ 公式 5.1.5

那么我们可得到 PID 的离散形式：

$$u(k)=K_p\left(err(k)+\frac{T}{T_i}\sum err(j)+\frac{T_D}{T}\big(err(k)-err(k-1)\big)\right)$$ 公式 5.1.6

又有$K_i=K_p\frac{T}{T_i}$ $K_d=K_p\frac{T_D}{T}$带入上式中有：

$$u(k)=K_p\left(err(k)+K_{i\sum err(j)}+K_d\big(err(k)-err(k-1)\big)\right)$$ 公式 5.1.7

最终可化简为：

$$u(k)=K_p err(k)+K_{i\sum err(k)}+K_d\big(err(k)-err(k-1)\big)$$ 公式 5.1.8

式中：

k：采样序号

$err(k)$：第次的误差

$u(k)$：输出值

K_p：比例常数

5.2 使用随机参数进行仿真

根据公式 x 构建加热系统，定义初始参数为 2.0，0.1，1.0，对加热系统进行仿真，得到的实际温度曲线如图 5.2 所示

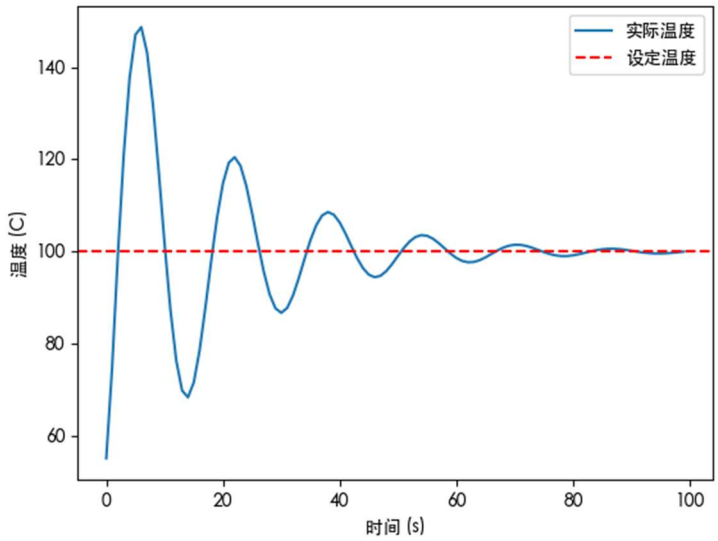

图 5.2：随机参数下的瞬态阶跃曲线

从图 5.2 中可以观察到，随机参数下的 PID 温度控制器存在过冲、超调等问题，实际温度曲线与设定温度曲线的拟合关系较差，且拟合时间过长。

计算瞬态阶跃性能指标，结果如表 5.2 所示

表 5.2：随机参数下的瞬态阶跃性能指标

参数名	参数值
上升时间	3.000s
过冲量	48.655℃
稳定时间	36.000s
稳态差值	0.0033℃

5.3 使用遗传算法整定 PID 参数

将加热系统相关参数送入 GA-PID 参数自整定算法中，得到该加热系统最佳的 PID 值如下表 5.3.1 所示。

表 5.3：最佳 PID 值

PID 参数	参数值
Kp	1.975
Ki	0.084
Kd	0.435

绘制系统的瞬态阶跃响应曲线如图 5.3.1 所示

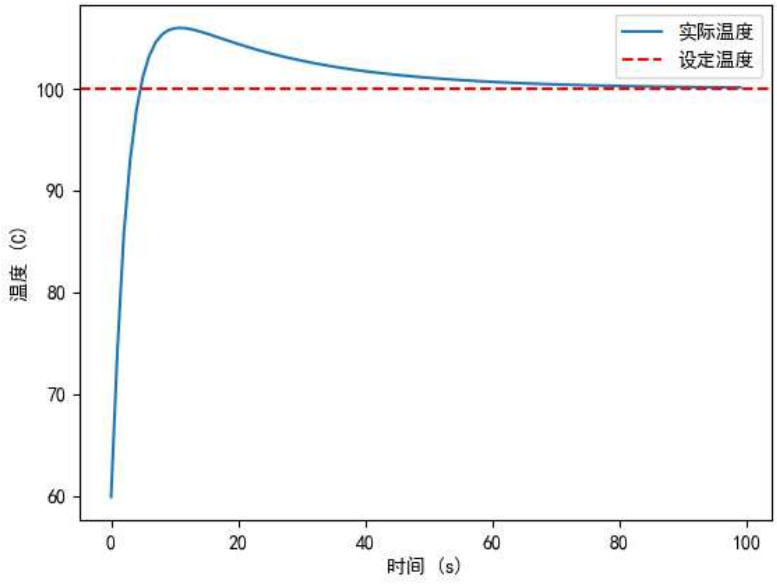

图 5.3.1：最佳 PID 值下系统的瞬态阶跃响应曲线

绘制适应度函数曲线如图 5.3.2 所示

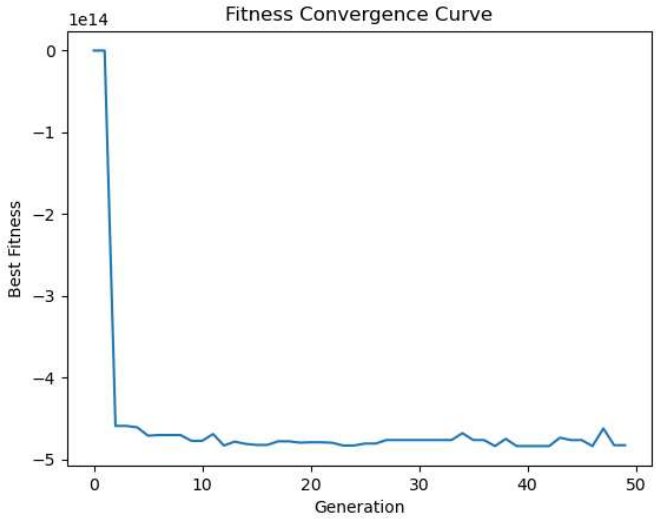

图 5.3.2：适应度函数曲线

计算系统的瞬态阶跃响应性能参数如表 5.3.2 所示

表 5.3.2：最佳 PID 值下系统的瞬态阶跃响应性能参数

参数名	参数值
上升时间	2.000s
过冲量	5.968℃
稳定时间	63.000s
稳态差值	0.0015℃

从上述图表中可以看出，使用遗传算法对 PID 控制系统进行参数自整定后，瞬态阶跃响应曲线拟合效果较好，性能参数有较大提升。

6 总结与展望

使用遗传算法进行 PID 参数自整定是一种有效的优化方法，通过模拟自然进化过程，寻找最优参数组合，提高控制系统的性能。未来的研究可以从改进适应度函数、优化遗传算法操作、引入混合优化方法[5]、实现实时在线整定、利用并行和分布式计算技术等方面进行，进一步提高 PID 控制器的优化效果和实际应用价值。

7 参考文献

[1]张立.基于串级 PID 控制算法的四旋翼无人机控制系统设计[J].信息技术与信息化,2023(1):100-103
[2]何进进,何志琴.无刷直流电机的模糊 PID 控制算法研究[J].机械与电子,2010,28(S1):80-82
[3]金强,石永康,吕玉龙,赵玉花.基于改进遗传算法的植保无人机姿态控制[J].农机化研究,2024,46(5):1-6
[4]冀炳晖,茅健,钱波.基于遗传算法-模糊 PID 的双喷头 FDM 型 3D 打印机温度控制方法[J].工程设计学报,2024,31(2):151-159
[5]曾彬洋.一种基于遗传算法的 PID 参数优化方法[J].当代化工研究,2024(2):187-190

8 附录

PID_Simulator.py

```
import matplotlib.pyplot as plt
import numpy as np


class PID:
    def __init__(self, Kp, Ki, Kd, setpoint, sample_time):
        self.Kp = Kp
        self.Ki = Ki
        self.Kd = Kd
        self.setpoint = setpoint
        self.sample_time = sample_time

        self._last_error = 0.0
        self._integral = 0.0

    def update(self, current_value):
        error = self.setpoint - current_value
        self._integral += error * self.sample_time
        derivative = (error - self._last_error) / self.sample_time

        output = self.Kp * error + self.Ki * self._integral + self.Kd
* derivative

        self._last_error = error

        return output


class HeatingSystem:
    def __init__(self, ambient_temp, heating_power, thermal_mass):
        self.ambient_temp = ambient_temp
        self.heating_power = heating_power
        self.thermal_mass = thermal_mass
        self.current_temp = ambient_temp

    def update(self, power_input, time_step):
        power_input = np.clip(power_input, -1e6, 1e6)
```

```
        temp_change = power_input * self.heating_power / self.thermal_mass * time_step
        self.current_temp += temp_change
        return self.current_temp


def simulate(Kp,Ki,Kd):
    # 仿真参数
    setpoint = 100  # 目标温度
    initial_temp = 20  # 初始温度
    ambient_temp = 20  # 环境温度
    heating_power = 100  # 加热功率
    thermal_mass = 500  # 热容
    sample_time = 1  # 采样时间
    simulation_time = 100  # 仿真时间

    # PID 参数
    # Kp = 0.47
    # Ki = 0.86
    # Kd = 0.87

    # 创建 PID 控制器和加热系统
    pid = PID(Kp, Ki, Kd, setpoint, sample_time)
    heating_system = HeatingSystem(ambient_temp, heating_power, thermal_mass)

    # 存储仿真结果
    times = []
    temperatures = []

    # 进行仿真
    for t in range(0, simulation_time, sample_time):
        power_input = pid.update(heating_system.current_temp)
        current_temp = heating_system.update(power_input, sample_time)

        times.append(t)
        temperatures.append(current_temp)

    # 绘制结果
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.plot(times, temperatures, label="实际温度")
    plt.axhline(y=setpoint, color='r', linestyle='--', label="设定温度")
    plt.xlabel('时间 (s)')
```

```
    plt.ylabel('温度 (C)')
    plt.legend()
    plt.savefig("PID_Simulator.png")
    plt.show()

    settling_time = None
    overshoot = None
    rise_time = None
    steady_state_error = abs(100 - temperatures[-1])

    # 计算上升时间
    for t, temp in zip(times, temperatures):
        if temp >= 100:
            rise_time = t
            break

    # 计算过冲
    overshoot = max(temperatures) - 100

    # 计算稳定时间（误差小于 2%设定值）
    for t, temp in reversed(list(zip(times, temperatures))):
        if abs(temp - 100) > 2:
            settling_time = t
            break

    settling_time = times[-1] - settling_time

    print(f"Rise Time: {rise_time} s")
    print(f"Overshoot: {overshoot} C")
    print(f"Settling Time: {settling_time} s")
    print(f"Steady State Error: {steady_state_error} C")

if __name__ == '__main__':
    simulate(0.47,0.86,0.87)
```

GA_PID.py

```
from PID_Simulator.PID_Simulator import PID
from PID_Simulator.PID_Simulator import HeatingSystem
from PID_Simulator.PID_Simulator import simulate
import numpy as np
import matplotlib.pyplot as plt

def fitness_function(params, setpoint, heating_system, sample_time,
simulation_time):
    Kp, Ki, Kd = params
    pid = PID(Kp, Ki, Kd, setpoint, sample_time)
    heating_system.current_temp = heating_system.ambient_temp

    error_sum = 0
    for t in range(0, simulation_time, sample_time):
        power_input = pid.update(heating_system.current_temp)
        current_temp = heating_system.update(power_input, sample_time)
        error = setpoint - current_temp

        # 限制误差值的范围，防止溢出
        error = np.clip(error, -1e6, 1e6)
        error_sum += error**2

    # 增加对大参数的惩罚，防止参数过大
    penalty = 0.1 * (Kp**2 + Ki**2 + Kd**2)

    return -(error_sum + penalty)  # 返回负的误差平方和加惩罚项



# 遗传算法参数
population_size = 100
generations = 500
mutation_rate = 0.1
crossover_rate = 0.7

# 参数范围
param_bounds = [(0.01, 5), (0, 0.5), (0, 0.5)]  # (Kp, Ki, Kd)的范围

# 初始化种群
population = np.random.rand(population_size, 3)  # 每个个体包含 3 个参数
```

```
（Kp、Ki、Kd）
for i in range(3):
    population[:, i] = population[:, i] * (param_bounds[i][1] -
param_bounds[i][0]) + param_bounds[i][0]

best_fitness_history = []

# 进行遗传算法
for generation in range(generations):
    fitness = []
    for individual in population:
        try:
            fitness_value = fitness_function(individual, setpoint=100,
heating_system=HeatingSystem(20, 1000, 500),
                                             sample_time=1,
simulation_time=500)
            if np.isnan(fitness_value):
                fitness_value = -np.inf
        except:
            fitness_value = -np.inf
        fitness.append(fitness_value)
    fitness = np.array(fitness)

    # 避免 NaN 和无穷值
    if np.any(np.isnan(fitness)) or np.any(np.isinf(fitness)):
        fitness[np.isnan(fitness) | np.isinf(fitness)] = -np.inf

    best_fitness = np.max(fitness)
    best_fitness_history.append(best_fitness)

    # 选择（轮盘赌选择）
    fitness_sum = np.sum(fitness)
    if fitness_sum == 0 or np.isinf(fitness_sum):
        probabilities = np.ones(population_size) / population_size
    else:
        probabilities = fitness / fitness_sum

    if np.any(np.isnan(probabilities)):
        probabilities = np.ones(population_size) / population_size

    selected_indices = np.random.choice(range(population_size),
size=population_size, p=probabilities)
    selected_population = population[selected_indices]
```

```
    # 交叉
    next_population = []
    for i in range(0, population_size, 2):
        parent1,       parent2       =       selected_population[i],
selected_population[i + 1]
        if np.random.rand() < crossover_rate:
            crossover_point = np.random.randint(1, 3)
            child1    =    np.concatenate((parent1[:crossover_point],
parent2[crossover_point:]))
            child2    =    np.concatenate((parent2[:crossover_point],
parent1[crossover_point:]))
        else:
            child1, child2 = parent1, parent2
        next_population.extend([child1, child2])

    # 变异
    next_population = np.array(next_population)
    for individual in next_population:
        if np.random.rand() < mutation_rate:
            mutation_point = np.random.randint(0, 3)
            individual[mutation_point] = np.random.rand() * (
                      param_bounds[mutation_point][1]               -
param_bounds[mutation_point][0]) + \
                                     param_bounds[mutation_point][0]
    population = next_population

# 找到最优参数
plt.plot(best_fitness_history)
plt.xlabel('Generation')
plt.ylabel('Best Fitness')
plt.title('Fitness Convergence Curve')
plt.show()


best_individual = population[np.argmax(fitness)]
best_Kp, best_Ki, best_Kd = best_individual
print(f"Best Kp: {best_Kp}, Best Ki: {best_Ki}, Best Kd: {best_Kd}")

pid = simulate(best_Kp, best_Ki, best_Kd)
```