基于灰狼算法的振荡浮子式波浪能参数优化设计

摘要

本文研究振荡浮子式波能装置的参数优化设计。本文首先通过受力分析建立动力学常微分方程，再使用四阶-五阶龙格库塔算法对浮子和振子的位移与速度进行求解，根据所得的速度对装置的平均输出功率进行计算，并建立以平均输出功率最大为目标的阻尼系数优化模型，最后使用灰狼智能算法寻优求解。

针对问题一，本文在只考虑垂荡运动条件下对浮子和振子受力分析，将**海水等效为弹簧阻尼系统**，构建 **双自由度受迫振动模型**，将方程**降阶**处理后使用 **四阶-五阶龙格库塔算法**分别对线性阻尼项方程和幂次阻尼项方程求解得到浮子和振子的垂荡位移与速度。

针对问题二，本文仍只考虑垂荡运动，根据问题一所求的速度计算波能装置平均输出功率，并建立目标为平均输出功率最大的**阻尼系数优化模型**，使用**灰狼智能算法**对模型求解，得到线性阻尼项时的最优阻尼系数为 **37672.2650**$N \cdot s/m$，此时最大输出功率为 **249.5598W**; 幂次阻尼项时的最优阻尼比例系数为 **38197.8583**$N \cdot s/m$ 和最优幂指数为 **0**, 此时最大输出功率为 **249.5730W**.

针对问题三，在考虑浮子同时进行垂荡和纵摇运动的情况下，对浮子和振子进行受力分析，建立了**基于刚体平面运动微分方程的五自由度能量转换模型**。解法采用四阶-五阶龙格库塔法，对二阶常微分方程组进行降阶处理，并解得浮子与振子的垂荡位移与速度和纵摇角位移与角速度。

针对问题四，本文建立了以系统输出功率最大为目标，阻尼器阻尼系数限制与相关运动学方程为约束条件的**优化模型**。首先通过**梯形积分法**，求解不同阻尼系数下的系统平均输出功率。采用灰狼算法，求解到最优直线阻尼系数为 **50857.9981** $N \cdot s/m$, 最优旋转阻尼系数为 **28640.3974**$N \cdot s/m$，此时的输出功率为 **289.157W**。

本文的优点为：1. 使用 3 自由度分析浮子的受力状态和运动情况，使用双自由度分析振子的受力状态和运动情况，简化计算过程，较为容易地得到各绝对量以及振子浮子之间运动的做功情况。2. 考虑振子和浮子质量的空间分布，求出了振子和浮子转动惯量的精确值，而非使用质点描述其受力情况，更符合实际。

关键词： 波能装置参数优化；多自由度受迫振动模型；灰狼算法；四阶-五阶龙格库塔算法

1 问题重述

1.1 问题背景

随着传统能源以惊人的速度枯竭，大气污染恶化，能源需求增加。为了解决目前的能源危机和污染问题，波浪能作为一种清洁、可持续的能源，成为了解决方案之一。为了利用海浪的能量, 世界各地都在进行研究工作。但各波浪能装置转换能量的效率仍不足以使波浪能得到大规模应用，因此如何设计波能装置使得能量转换功率最大成为研究的焦点。

波能装置通常由能量俘获系统和能量转换系统 (PTO 系统) 两个部分组成，其中振荡浮子式波浪能装置具有能量转换效率高，易于建造的优点 [1]。其结构包含浮子、振子和 PTO 系统, 其中 PTO 系统由弹簧和阻尼器组成。当波浪作用于浮子时, 浮子俘获波浪能并将振荡传递给位于浮子内部的振子, 使波浪能转化为机械能。振子连接 PTO 系统, 通过 PTO 中的阻尼器做功, 将机械能转化为电能。

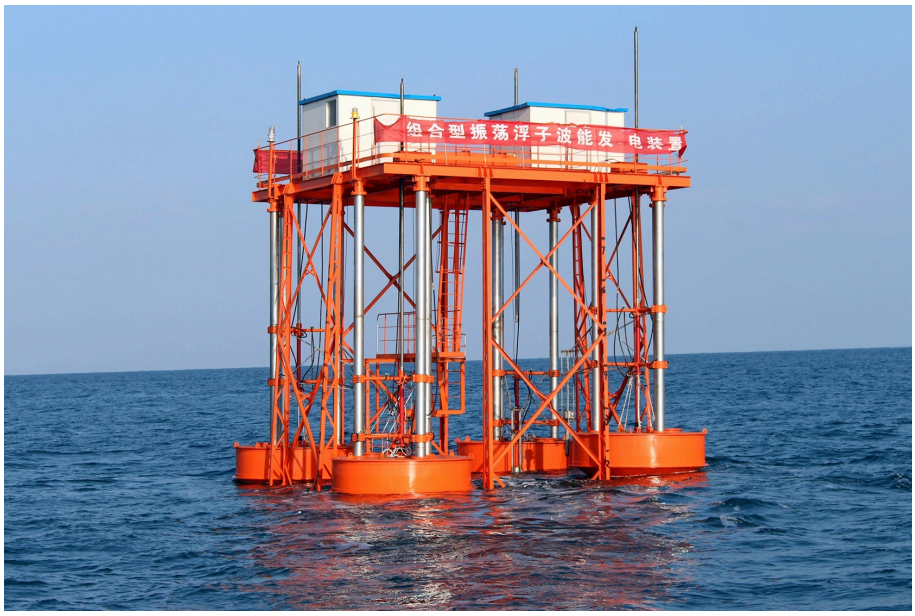

图 1-1　振荡浮子式波浪能装置

1.2 问题提出

本文将针对振荡浮子式波浪能装置的两种设计结构, 通过分析不同阻尼系数下的振荡运动，分别做出能量转换功率最大的阻尼系数优化设计。具体问题如下：

问题一: 只考虑垂荡运动，已知波浪频率为 $1.4005s^{-1}$, 建立数学模型描述浮子和振子在阻尼系数为常量或阻尼系数为浮子振子相对速度幂函数时，浮子和振子分别在前 40 个周期内，每间隔 0.2s 垂荡位移和速度。

问题二: 只考虑垂荡运动，已知波浪频率为 $2.2143s^{-1}$, 建立优化模型对问题一中两种情况分别求解最优阻尼系数，并给出对应最大功率。

问题三: 只考虑垂荡和纵摇运动，已知波浪频率为 $1.7152s^{-1}$, 建立模型描述当两种阻尼器阻尼系数已知且为常量时, 浮子和振子分别在前 40 个周期内，每间隔 0.2s 垂荡位移、速度、纵摇角位移和角速度。

问题四: 只考虑垂荡和纵摇运动，已知波浪频率为 $1.9806s^{-1}$, 建立优化模型确定两种阻尼器的最优阻尼系数，并给出对应最大功率。

2 基本假设与符号说明

2.1 基本假设

为简化模型，本文提出以下几点假设：

1. 假设浮子、振子均为刚体，在运动过程中不会发生形变；
2. 假设阻尼器做功全部转换为电能；
3. 假设垂直弹簧只发生纵向形变，不发生扭转、弯曲等形变；扭转弹簧只发生扭转形变，不发生纵向、弯曲等形变。

2.2 符号说明

表 2-1　符号说明表

符号	含义	单位
m_f	浮子质量	kg
m_z	振子质量	kg
m_δ	惯性附加质量	kg
y_f	浮子位移	m
y_z	振子位移	m
l	弹簧原长	m
k	弹簧刚性系数	N/m
h_{cw}	静水时圆柱体没入水面高度	m
h_c	锥体高度	m
ρ_w	海水密度	kg/m^3

注：表中未说明的符号以首次出现处为准

3 问题一的模型与求解

3.1 问题一的分析

问题一要求在只考虑垂荡运动下，对波能装置中的浮子和振子的运动进行描述。本文首先对该物理情形建立一维坐标系，并使中轴底座中心为原点从而简化模型，随后将海水对浮子的力等效为弹簧阻尼系统，建立了双自由度受迫振动模型并通过受力分析确定了动力学方程。由于需要考虑线性阻尼和非线性阻尼两种情况，本文对动力学方程进行降阶处理，后使用四阶-五阶龙格库塔算法对模型进行求解，量化了线性阻尼和非线性阻尼影响下浮子和振子的运动情况。

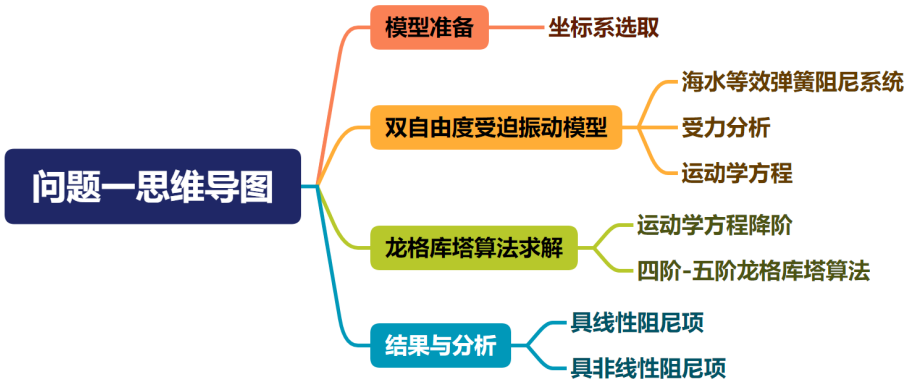

图 3-1　问题一思维导图

3.2 模型准备

3.2.1 装置结构

第一种设计结构如图 3-1 所示，中轴一端固定于浮子顶面中心, 一端固定于中轴底座中心。此时 PTO 系统包含连接振子与中轴底座的一个弹簧和一个直线阻尼器, 且直线阻尼器所产生的力等于浮子和振子的相对速度乘以阻尼系数。

3.2.2 坐标系选取

对于只考虑垂荡运动的振荡浮子式波能装置，本文对其建立一维坐标系，如下图所示。以垂直海平面竖直向上为 z 轴正方向，以波能装置静水时中心底座中心为原点，分析浮子与振子在 z 轴上的位移与速度。

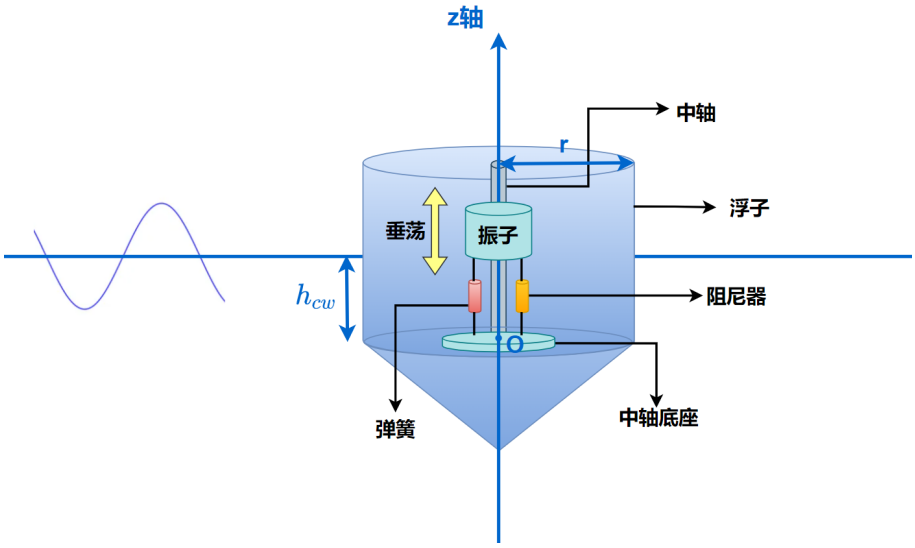

图 3-2　装置结构与坐标系确定

3.3 双自由度受迫振动系统

当波浪作用于浮子时，浮子将受到波浪激励力、兴波阻尼力、静水恢复力和附加惯性力。其中本文将兴波阻尼力看作海水阻尼器，静水恢复力看作海水弹簧，因此浮子与海水构成了一套质量弹簧阻尼系统，结合浮子内部的振子 PTO 系统，形成了双自由度受迫振动系统 [2]。

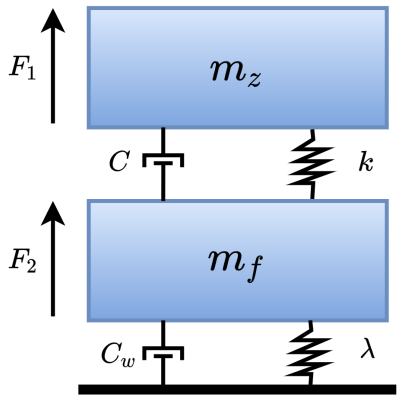

图 3-3　双自由度受迫振动系统示意图

图中 m_f 为浮子，m_z 为振子，对两者进行受力分析，可得其运动方程.

3.3.1 控制方程

根据牛顿第二定律，建立动力学方程如下：

$$\begin{cases} m_z y_z'' = -k(y_z - y_f - l) - c(y_z' - y_f') - m_z g \\ (m_f + m_\delta) y_f'' = k(y_z - y_f - l) + c(y_z' - y_f') - c_w y_f' - m_f g + \rho_w V_w g + f\cos\omega t \end{cases} \tag{3.1}$$

其中

$$V_w = \pi(\frac{1}{3}r^2 h_c + h_{cw} - y_f)$$

式中 h_c 为锥体高度；h_{cw} 为静水条件下圆柱体没入海面的深度；ρ_w 为海水密度；V_w 为浮子没入海面的体积。

3.3.2 初始条件

初始条件下，浮子与振子均在静水中静止，因此位移与速度初值如下：

$$\begin{cases} y_f(0) = 0, y_f'(0) = 0 \\ y_z(0) = l_{u0}, y_z'(0) = 0 \end{cases} \tag{3.2}$$

3.4 龙格库塔算法求解模型

3.4.1 四阶-五阶 Runge-Kutta 算法原理

数值分析中，龙格-库塔法 (Runge-Kutta methods) 是用于非线性常微分方程的解的重要的一类隐式或显式迭代法。其中四阶-五阶龙格库塔法十分常用，它用 4 阶方法提供候选解，5 阶方法控制误差，是一种自适应步长 (变步长) 的常微分方程数值解法，其整体截断误差为 $(\Delta x)^5$[3]。

本文在此采用龙格库塔算法的隐式迭代格式，具有以下形式：

$$y_{n+1} = y_n + \sum_{i=1}^{s} b_i k_i \tag{3.3}$$

其中

$$k_i = f\left(t_n + c_i h, y_n + h\sum_{j=1}^{s} a_{ij} k_j\right), \quad i = 1, \ldots, s \tag{3.4}$$

由于龙格库塔算法只适用于一阶方程，因此在使用前需要对方程进行降阶

3.4.2 方程降阶处理

令 $x_z = y_z^{'}$,$x_f = y_f^{'}$, 因此可以将模型中的两个二阶微分方程降阶成为四个一阶微分方程，如下式：

$$\begin{cases} y_f^{'} = x_f \\ y_z^{'} = x_z \\ m_z x_z^{'} = -k(y_z - y_f - l) - c(x_z - x_f) - m_z g \\ (m_f + m_g) x_f^{'} = k(y_z - y_f - l) + c(x_z - x_f) - c_w x_f - m_f g + \rho_w V_w g + f\cos\omega t \end{cases} \tag{3.5}$$

此时可使用四阶-五阶龙格库塔算法对上述方程组在阻尼系数为常数情况和阻尼系数为浮子与振子相对速度幂函数情况下分别求解。

3.4.3 具线性阻尼项方程求解

当阻尼系数为常数时，将 $c = 10000N \cdot s/m$ 代入式 (3.5), 算法求解得到结果如下表所示：

表 3-1　问题一线性阻尼项结果

运动参数	10s	20s	40s	60s	100s
浮子位移 (m)	-0.1905	-0.5904	0.2854	-0.3144	-0.0836
振子位移 (m)	-0.2114	-0.6341	0.2966	-0.3314	-0.0840
浮子速度 (m/s)	-0.6398	-0.2403	0.3139	-0.4789	-0.6038
振子速度 (m/s)	-0.6935	-0.2709	0.3337	-0.5151	-0.6432

3.4.4 具非线性阻尼项方程求解

当阻尼系数为浮子与振子相对速度幂函数时，由题可知，$c = 10000(y_f^{'} - y_z^{'})^{0.5}$. 将此式代入式 (3.5)，算法求解得到结果如下：

表 3-2　问题一非线性阻尼项结果

运动参数	10s	20s	40s	60s	100s
浮子位移 (m)	-0.2061	-0.6116	0.2687	-0.3273	-0.0882
振子位移 (m)	-0.2347	-0.6602	0.2799	-0.3491	-0.0939
浮子速度 (m/s)	-0.6511	-0.2565	0.2970	-0.4921	-0.6104
振子速度 (m/s)	-0.6999	-0.2687	0.3122	-0.5213	-0.6469

3.4.5 结果分析

将线性阻尼系数和幂次阻尼系数两种情况下的浮子振子随着时间的位移变化进行比较，结果如下图所示：

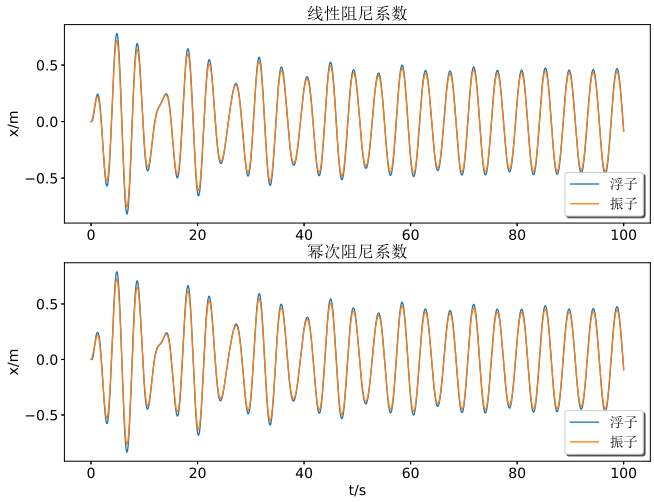

图 3-4　两种阻尼系数下浮子振子位移随时间变化

将线性阻尼系数和幂次阻尼系数两种情况下的浮子振子随着时间的速度变化进行比较，结果如下图所示：

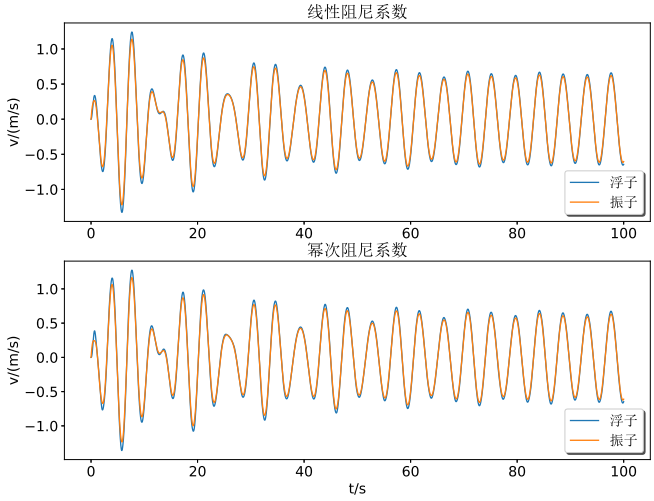

图 3-5　两种阻尼系数下浮子振子速度随时间变化

从上述结果中可以看出，线性阻尼条件下的振子位移略大于幂次阻尼条件下的振子位移，同时线性阻尼条件下的振子速度也略大于幂次阻尼条件下的速度。

4 问题二的模型与求解

4.1 问题二的分析

问题二在问题一的基础上要求得到最优阻尼系数使得波能装置达到最大平均输出功率，本文首先提出波能装置瞬时功率计算方法，再将瞬时功率在一段时间内积分取平均，得到平均输出功率。随后建立优化目标为平均输出功率最大的优化模型，并分线性阻尼和非线性阻尼两种情况调节约束，最后使用灰狼算法寻优求解。

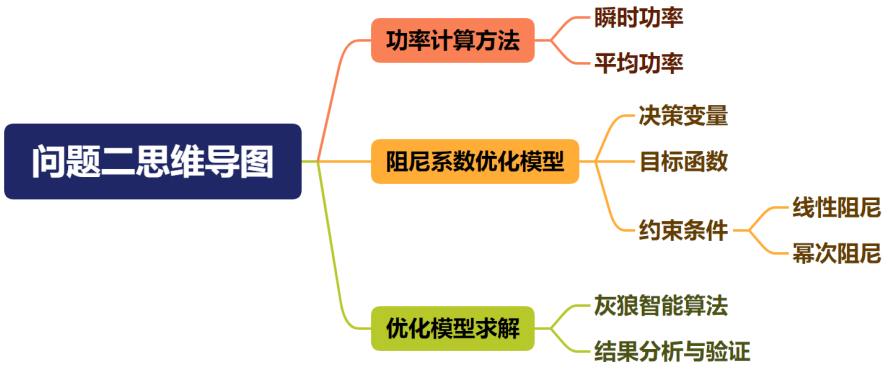

图 4-1　问题二思维导图

4.2 阻尼系数优化模型

4.2.1 功率计算方法

能量转换系统的发电是通过振子运动克服 PTO 系统中直线阻尼力做功实现，因此，波能装置的瞬时输出功率可由式 (4.1) 表示 [4]。

$$P_t = F(t) \cdot v(t) = c \cdot v^2(t) \tag{4.1}$$

得到瞬时功率后，对一段时间的瞬时功率积分并取平均值，从而得到平均输出功率，计算公式为：

$$\bar{P} = \frac{1}{T}\int_0^T P_t dt = \frac{1}{T}\left[\frac{(P_1 + P_n)\Delta t}{2} + \sum_{i=2}^{n-1} P_i \Delta t\right] \tag{4.2}$$

式中 Δt 为时间步长,T 为计算时长,n 为采样点总数。

4.2.2 决策变量

优化模型的决策变量是阻尼系数 c. 通过调节阻尼系数, 可改变波能装置的输出功率。

4.2.3 目标函数

由于本问要求使 PTO 系统的平均输出功率最大，因此建立优化目标函数为：

$$\max \bar{P} \tag{4.3}$$

4.2.4 约束条件

(1) 当阻尼系数为常量时

根据题目要求，此时阻尼系数 c 取值不得超过区间 [0,100000]. 此时优化模型约束条件为

$$0 \leqslant c \leqslant 100000 \tag{4.4}$$

(2) 当阻尼系数为浮子振子相对速度绝对值的幂函数时

根据题目要求，此时阻尼系数 c 与浮子振子相对速度绝对值的幂函数成正比. 阻尼系数可表示为：

$$c = \eta |y_f^{'} - y_z^{'}|^{\alpha} \tag{4.5}$$

此时比例系数 η 取值不得超过 [0,100000] 区间, 幂函数指数 α 在区间 [0,1] 之间取值. 因此优化模型约束条件为

$$\begin{cases} 0 \leqslant \eta \leqslant 100000 \\ 0 \leqslant \alpha \leqslant 1 \end{cases} \tag{4.6}$$

4.2.5 模型综述

综上所述, 结合问题一受迫振动模型, 建立输出功率最大的阻尼系数优化模型.

$$c = \underset{c}{arg\ max} \bar{P}$$

$$s.t. \begin{cases} \text{控制方程}: \begin{cases} m_z y_z^{''} = -k(y_z - y_f - l) - c(y_z^{'} - y_f^{'}) - m_z g \\ (m_f + m_\delta) y_f^{''} = k(y_z - y_f - l) + c(y_z^{'} - y_f^{'}) - c_w y_f^{'} \\ \qquad\qquad -m_f g + \rho_w V_w g + f\cos\omega t \end{cases} \\ \text{初始条件}: \begin{cases} y_f(0) = 0, y_f'(0) = 0 \\ y_z(0) = l_{u0}, y_z'(0) = 0 \end{cases} \\ \text{系数范围}: \begin{cases} \text{阻尼项为线性：} \quad 0 \leqslant c \leqslant 100000 \\ \text{阻尼项非线性：} \begin{cases} 0 \leqslant \eta \leqslant 100000 \\ 0 \leqslant \alpha \leqslant 1 \end{cases} \end{cases} \end{cases} \tag{4.7}$$

4.3 灰狼算法求解优化模型

针对上述优化问题，本文采用灰狼算法 (GWO) 来进行求解。该算法通过模拟灰狼群体捕食行为，基于狼群群体协作的机制来达到优化的目的 [5]。

4.3.1 灰狼算法原理

灰狼算法（GWO）是一种群体智能算法，灵感来源于灰狼的领导层及和狩猎机制，模拟了灰狼寻找猎物、包围猎物、攻击猎物的行为。

灰狼在狩猎过程中保留三个最佳解决方案，最适解定义为 α，第二最优解为 β，第三最优解为 δ，这三个解即为狼群的领导层级，领导整个狩猎过程，而其余狼只 ω 则跟随 α、β、δ 三只狼。

4.3.2 灰狼算法步骤

- **Step1: 包围猎物**

灰狼搜索猎物时会逐渐地接近猎物并包围它，包围过程中计算灰狼个体与猎物之间的距离 D, 并根据个体的距离进行位置更新。

- **Step2: 狩猎**

狩猎过程首先确定三只领导层级灰狼的位置与其他个体之间的距离, 其余灰狼 ω 则跟随领导层级灰狼更新位置，从而确定其移动步长

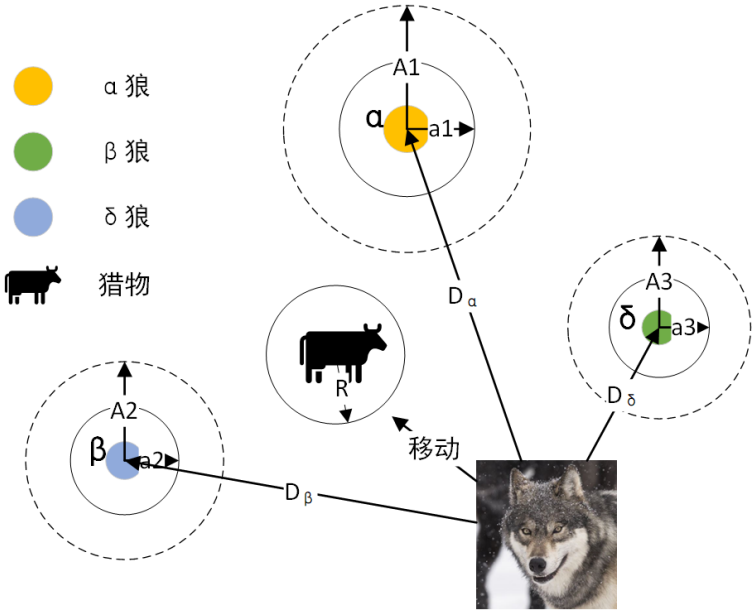

图 4-2　灰狼位置更新示意图

- **Step3: 攻击猎物**

在猎物停止移动时，灰狼采取攻击行为。为了模拟逼近猎物，收敛因子 a 的数值不断减小，因此灰狼位置系数向量 A 的数值波动也随之收敛。当 A 的值位于区间内时，灰狼的下一位置可以位于其当前位置和猎物位置之间的任意位置。当 $A < 1$ 时，狼群向猎物发起攻击（陷入局部最优）。

4.3.3 灰狼算法流程图

将上述灰狼狩猎过程仿生带入编程中，算法流程图如下：

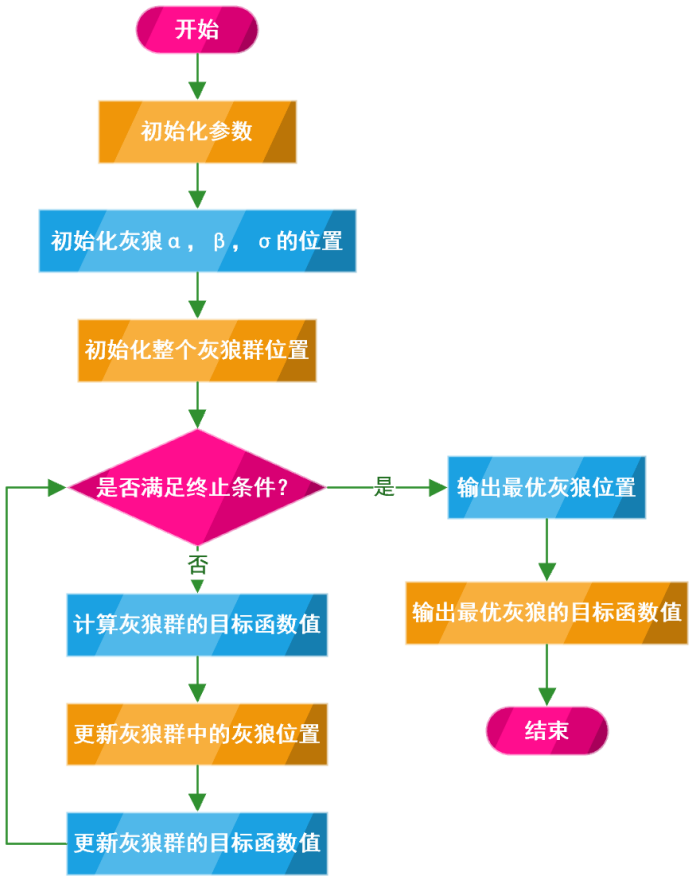

图 4-3　灰狼算法流程图

4.4 求解结果

4.4.1 线性阻尼结果

当阻尼系数为常量时，使用灰狼算法对其进行寻优，解得此时的最优参数以及对应最大功率如下：

表 4-1　问题二线性阻尼项结果

参数项	数值
最优系数 (单位:$N \cdot s/m$)	37672.2650
最优功率 (单位:W)	249.5598

4.4.2 幂次阻尼结果

当阻尼系数为相对速度绝对值的幂函数时，使用灰狼算法对其比例系数和幂指数进行寻优，解得此时最优参数及对应最大功率如下表：

表 4-2　问题二幂次阻尼项结果

参数项	数值
最优系数 (单位:$N \cdot s/m$)	38197.8583
最优指数	0
最优功率 (单位:W)	249.5730

4.5 结果分析

本文对于幂次阻尼系数最优指数结果为 0 进行了进一步的结果验证。

如图 4-3 所示，当幂指数为 0 时，蓝色线为发电功率随阻尼系数变化曲线，红色点为灰狼智能算法确定的最高点，与曲线最高点吻合较好，模型准确率较高。

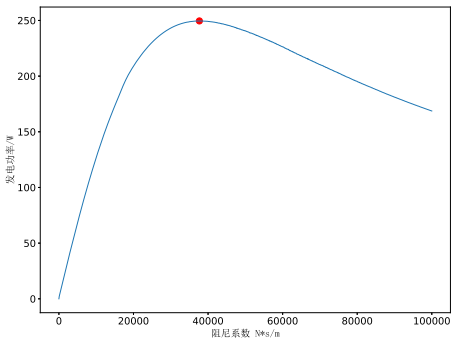

图 4-4　灰狼寻优验证曲线

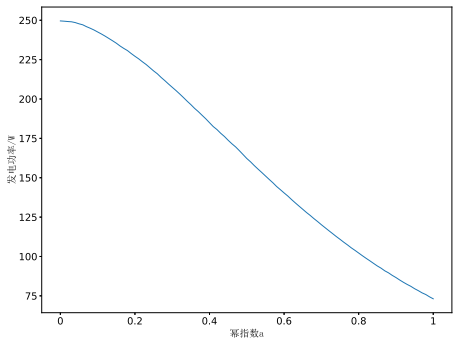

图 4-5　功率随幂指数变化曲线

由于线性解与幂次解出现同解问题，本问固定线性最优系数结果，绘制发电功率随幂指数变化的曲线如图 4-4. 由图可见，功率随着幂指数增加而下降，因此幂指数为 0 确为最优解。从而可以得到结论，**线性阻尼项条件下输出功率更大**。

可以得到阻尼系数和幂指数对发电功率的影响如下图所示：

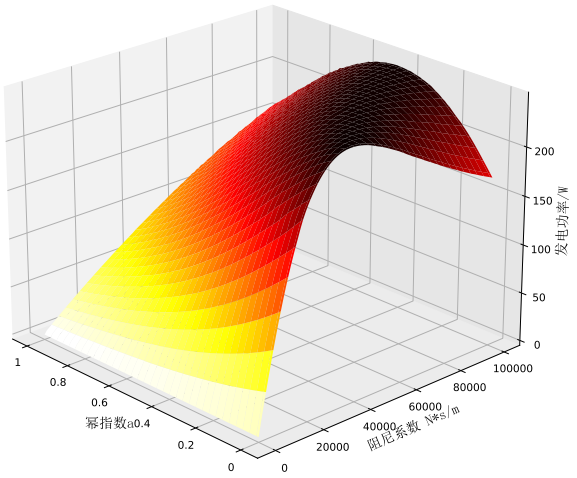

图 4-6　功率随幂指数和阻尼系数变化图

5 问题三的模型与求解

5.1 问题三的分析

问题三的 PTO 系统额外安装了旋转阻尼器和旋转弹簧，在考虑垂荡运动的基础上还需要分析纵摇运动对系统运动和做功的影响。根据刚体平面运动定律对浮子和振子的受力和运动进行分界，使用 x、y、θ_1，三个自由度描述浮子的运动状态，而使用 r、θ_2，两个自由度描述振子的运动状态，从而建立五自由度能量转换模型，并通过龙格库塔法求解微分方程组，从而得到振子相对浮子运动的做功情况以及相关参量。

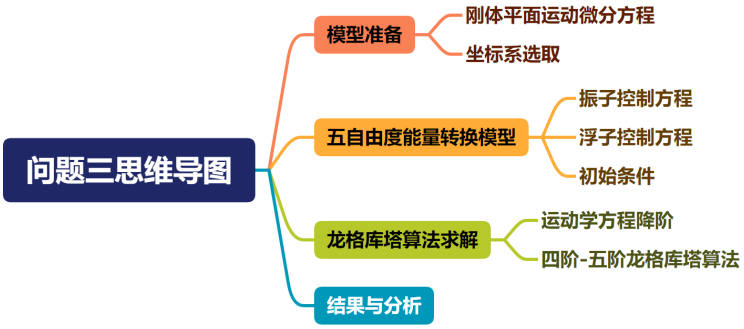

图 5-1　问题三思维导图

5.2 模型准备

5.2.1 刚体平面运动微分方程

刚体的平面运动可分解为随基点的平移加上绕基点的转动 [6]。

设刚体在 Oxy 平面内作平面运动。取刚体质点 C 为基点，其位置坐标为 (x_c, y_c), 刚体绕质心 C 的转动由与刚体固连的直线 CD 和 x 轴间的夹角 φ 来确定。由质心运动定

理和质点系对于质心的动量矩定理，得

$$\begin{cases} ma_c = \sum F_i^{(e)} \\ \frac{d}{dt}(J_{c\omega}) = J_{c\alpha} = \sum M_c(F_i^{(e)}) \end{cases} \tag{5.1}$$

其中，m 为刚体得质量，a_c 为质心的加速度，J_c 为刚体对通过质心 C 且与 Oxy 平面垂直的轴的转动惯量。$\alpha = \frac{d\omega}{dt}$ 为刚体的角加速度。式 (3.1) 又可写为：

$$\begin{cases} m\frac{d^2 r_c}{d_i^2} = \sum F_i^{(e)} \\ J_c\frac{d^2\varphi}{dt^2} = \sum M_c(F_i^{(e)}) \end{cases} \tag{5.2}$$

5.2.2 装置结构

第二种设计结构如图 5-2 左图所示，中轴只一端固定于中轴底座中心, 轴可绕转轴转动。此时 PTO 系统在第一种设计上新增了扭转弹簧和旋转阻尼器, 直线阻尼器与旋转阻尼器同时做功。相同地, 旋转阻尼器的阻尼力等于浮子和振子相对角速度乘以阻尼系数。此时浮子在波浪中做摇荡运动, 即垂荡运动与纵摇运动的叠加。

5.2.3 坐标系选取

对于只考虑垂荡和纵摇运动的振荡浮子式波能装置，本文取中轴底座的中心作为原点，对浮子建立了绝对坐标系 Oyz，对振子建立了以垂直海平面为极轴的极坐标系，如图 5-2 右图所示。在此建系条件下，本文通过绝对坐标系中 z,y 以及纵摇角度 θ_f 来描述浮子的运动，通过极坐标中的 r 和 θ_z 来描述振子的运动。因此建立五自由度运动学模型 [7]。

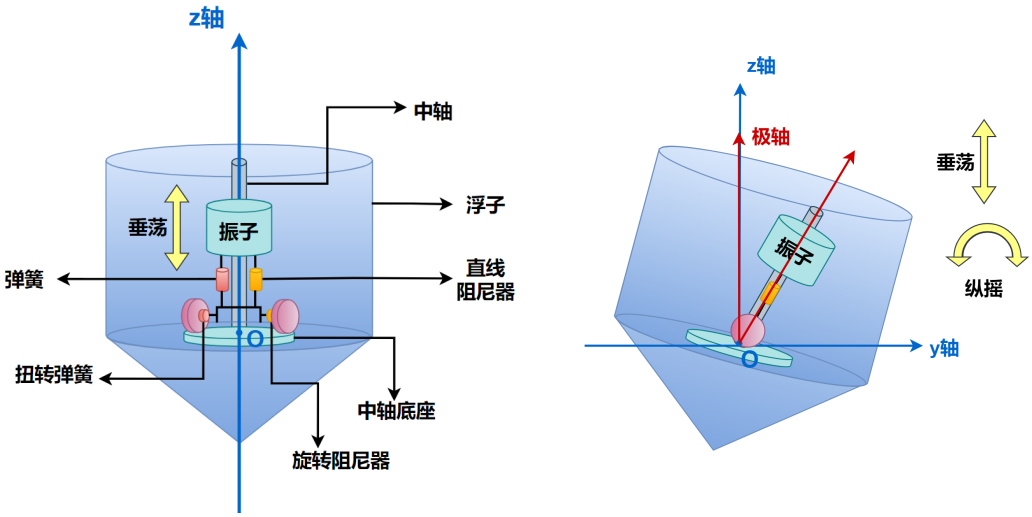

图 5-2　摇荡装置结构和坐标系确定

5.3 五自由度能量转换模型

当考虑浮子和振子做垂荡和纵摇运动时，本文分别对浮子和振子进行受力分析，并将运动分解为垂荡运动和纵摇运动分别建立动力学方程 [8]。

5.3.1 振子控制方程

$$\begin{cases} m_z z'' = -k(r-l) - cr' - m_z g\cos\theta_v + m_v\left(\theta_z'\right)^2 r \\ J_z\theta_z'' = -k_r\left(\theta_z - \theta_f\right) - c_r\left(\theta_z' - \theta_f'\right) + m_z g r\sin\theta_z \end{cases} \tag{5.3}$$

其中，r 为振子相对浮子底座的距离；θ_z 振子相对静水的纵摇角位移；k_r 为扭转弹簧的刚度；c_r 为扭转阻尼系数。

J_z 为振子相对连接处的转动惯量, 计算方式如下：

$$J_z = \rho_z \iiint_{D_z}\left(y^2+z^2\right)dz \tag{5.4}$$

其中 $V_z = \frac{\pi}{8}, \rho_z = \frac{8}{\pi}m_z, D_z = \left\{(x,y,z)\mid y^2+z^2 \le 0.25, 0.5 \le z \le r+0.5\right\}$.

5.3.2 浮子控制方程

当浮子呈某一角度倾斜于水中做垂荡运动时，其排水的体积可计算如下 [9]：

$$V_\omega = (\frac{4}{15} + h_0 - y)\pi + \frac{|\tan\vartheta|}{1+\tan^2\vartheta}\pi + \pi|\tan\vartheta| \tag{5.5}$$

$$\begin{cases} \left(m_f+m_a\right)y'' = f\cos\omega t - c_w y' + \rho g V_w + k(r-l)\cos\theta_z + cr'\cos\theta_z \\ \left(m_f+m_a\right)x'' = -c_w x' + k(r-l)\sin\theta_z + cr'\sin\theta_z \\ \left(J_f+J_a\right)\theta_f'' = L\cos\omega t - c_{cr}\theta_f - k_h\theta_f + k_r\left(\theta_z-\theta_f\right) + c_r\left(\theta_z'-\theta_f'\right) \end{cases} \tag{5.6}$$

其中，θ_f 为浮子相对静水的纵摇角位移；k_h 为静水恢复力矩系数；c_{cr} 为纵摇行波阻尼系数；c_ω 为垂荡兴波阻尼系数;J_a 为浮子相对连接处的附加转动惯量.

J_f 为浮子相对连接处的转动惯量, 计算方式如下：

$$J_f = \rho_f \iint_{\Sigma_1+\Sigma_2+\Sigma_3}\left(y^2+z^2\right)dS \tag{5.7}$$

其中，Σ_1,Σ_2,Σ_3 代表将转动分为圆柱表面、圆柱顶面和圆锥表面。[10]

因此 $\Sigma_1 = \left\{(x,y,z)\mid x^2+y^2=1, 0\le z\le 3\right\}$;

$\Sigma_2 = \left\{(x,y,z)\mid x^2+y^2\le 1, z=3\right\}$;

$\Sigma_3 = \left\{(x,y,z)\mid z=\frac{4}{5}\sqrt{x^2+y^2}-\frac{4}{5}, -\frac{4}{5}\le z\le 0\right\}$.

且 $S_f = 6\pi+\pi+\sqrt{1.64}\pi = (7+\sqrt{1.64})\pi, \rho_f = \frac{1+\sqrt[v]{\pi}.64}{}m_v$

5.3.3 初始条件

初始条件下，浮子振子均静止于静水中，因此初值条件如下：

$$\begin{cases} y=0, y'(0)=0 \\ x=0, x'(0)=0 \\ r(0)=l_{u0}, r'(0)=0 \\ \theta_f=0, \theta_f'(0)=0 \\ \theta_z=0, \theta_z'(0)=0 \end{cases} \tag{5.8}$$

5.4 龙格-库塔算法求解模型

对于加入纵摇运动后的运动方程，本文仍沿用问题一中使用过的四阶-五阶龙格库塔算法对微分方程进行求解。

5.4.1 求解结果

使用龙格库塔算法求解后得到浮子运动参量如下表：

表 5-1　问题三浮子运动参量

运动参量	10s	20s	40s	60s	100s
垂荡位移 (m)	-0.5283	-0.7048	0.3694	-0.3207	-0.0502
垂荡速度 (m/s)	0.9698	-0.2693	0.7576	-0.7218	-0.9467
纵摇角位移	0.0004	0.0004	-0.0010	0.0008	0.0003
纵摇角速度 (s-1)	-0.0010	0.0002	-0.0004	0.0007	0.0004

使用龙格库塔算法求解后得到振子运动参量如下表：

表 5-2　问题三振子运动参量

运动参量	10s	20s	40s	60s	100s
垂荡位移 (m)	-0.3968	-0.5703	0.5946	-0.1395	0.1593
垂荡速度 (m/s)	1.0382	-0.3190	0.8450	-0.7993	-1.0365
纵摇角位移	0.0004	0.0004	-0.0010	0.0008	0.0003
纵摇角速度 (s-1)	-0.0010	0.0002	-0.0004	0.0007	0.0004

6 问题四的求解与分析

6.1 问题四的分析

问题四在问题三的基础上要求求解最优阻尼系数使得波能装置达到最大平均输出功率，本文将功率分为直线阻尼功率和旋转阻尼功率，分别求得各自的瞬时功率，再将瞬时功率相加在一段时间内积分取平均，得到平均输出功率随后建立优化目标为平均输出功率最大的优化模型，最后使用灰狼算法寻优求解。

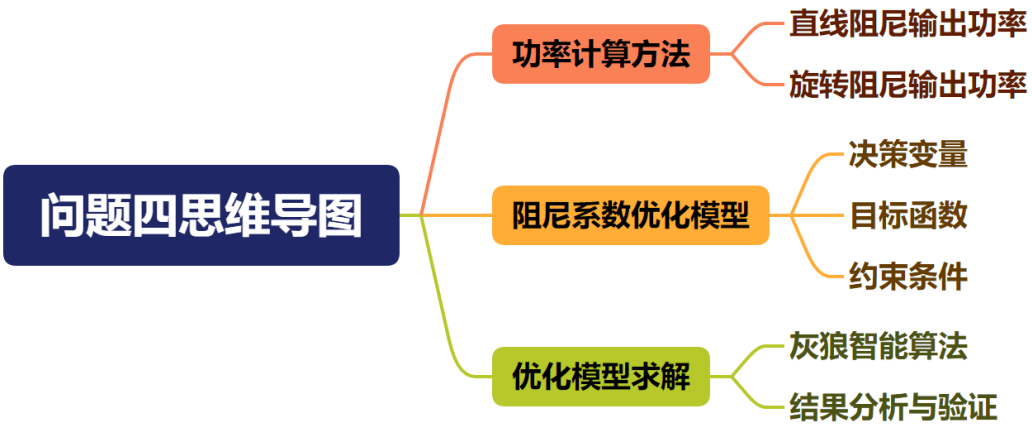

图 6-1　问题四思维导图

6.2 阻尼系数优化模型

6.2.1 功率计算方法

能量转换系统的发电是通过振子运动克服 PTO 系统中直线阻尼力和旋转阻尼力共同做功实现，因此，波能装置的瞬时输出功率分为直线阻尼输出功率 P_1 和旋转阻尼输出功率 P_2, 如式 (6.1) 表示。

$$\begin{aligned} P_1 &= c\,(r')^2 \\ P_2 &= c_r\left(\theta'_z - \theta'_f\right)^2 \end{aligned} \tag{6.1}$$

得到瞬时功率后，对一段时间的瞬时功率积分并取平均值，从而得到平均输出功率，计算公式为：

$$\bar{P} = \frac{1}{T}\int_0^T \left[P_1(t) + P_2(t)\right] dt \tag{6.2}$$

上式沿用问题二中的梯形法进行简化求解 [9]。

6.2.2 决策变量

优化模型的决策变量是直线阻尼系数 c 和扭转阻尼系数 c_r. 通过调节阻尼系数, 可改变波能装置的输出功率。

6.2.3 目标函数

由于本问要求使 PTO 系统的平均输出功率最大，因此建立优化目标函数为：

$$\max \bar{P} \tag{6.3}$$

6.2.4 约束条件

根据题目要求，此时直线阻尼系数 c 和扭转阻尼系数 c_r 在区间 [0,100000] 内取值. 此时优化模型约束条件为

$$\begin{cases} 0 \leqslant c \leqslant 100000 \\ 0 \leqslant c_r \leqslant 100000 \end{cases} \tag{6.4}$$

6.2.5 模型综述

综上所述, 结合问题一受迫振动模型, 建立输出功率最大的阻尼系数优化模型.

$$(c, c_r) = \underset{c,c_r}{arg\ max} \bar{P}$$

$$s.t. \begin{cases} \text{振子控制方程}: \begin{cases} m_z r'' = -k(r-l) - cr' - m_z g\cos\theta_z + m_z\left(\theta_z'\right)^2 r \\ J_z\theta_z'' = -k_r\left(\theta_z - \theta_f\right) - c_r\left(\theta_z' - \theta_f'\right) + m_z g r\sin\theta_z \end{cases} \\ \text{浮子控制方程}: \begin{cases} \left(m_f + m_a\right) y'' = f\cos\omega t - c_w y' + \rho g V_w + k(r-l)\cos\theta_v + cr'\cos\theta_v \\ \left(m_f + m_a\right) x'' = -c_w x' + k(r-l)\sin\theta_v + cr'\sin\theta_v \\ \left(J_f + J_a\right)\theta_f'' = L\cos\omega t - c_{cr}\theta_f - k_h\theta_f + k_r\left(\theta_v - \theta_f\right) + c_r\left(\theta_v' - \theta_f'\right) \end{cases} \\ \text{初始条件}: \begin{cases} y = 0, y'(0) = 0 \\ x = 0, x'(0) = 0 \\ r(0) = l_{u0}, r'(0) = 0 \\ \theta_f = 0, \theta_f'(0) = 0 \\ \theta_z = 0, \theta_z'(0) = 0 \end{cases} \\ \text{系数范围}: \begin{cases} 0 \leqslant c \leqslant 100000 \\ 0 \leqslant c_r \leqslant 100000 \end{cases} \end{cases} \tag{6.5}$$

6.3 灰狼算法求解优化模型

对于上述优化模型，本文依旧沿用问题二中使用的灰狼算法对其进行寻优。

6.3.1 求解结果

对上述模型使用灰狼算法寻优求得结果如下表所示：

表 6-1　问题四寻优结果

参数项	数值
最优直线阻尼系数 (单位:$N \cdot s/m$)	50857.9981
最优扭力阻尼系数 (单位:$N \cdot s/m$)	28640.3974
最大平均输出功率 (单位:W)	289.1578

6.3.2 结果分析

输出功率随时间变化曲线如下，曲线呈振荡形，且振幅先逐渐增大，后缓慢减小。

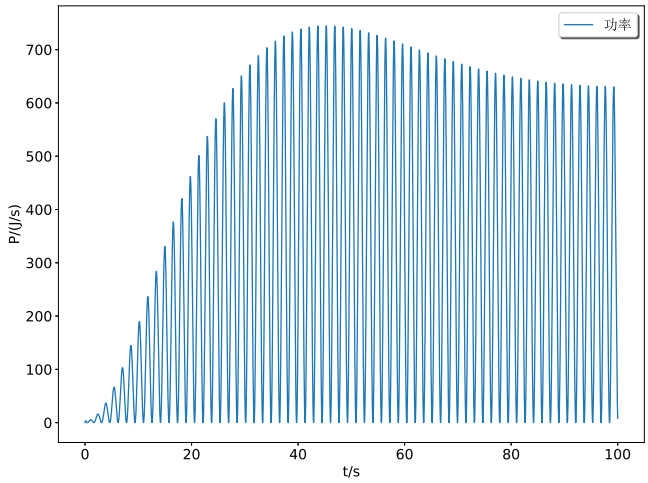

图 6-2　输出功率随时间变化

7 模型优缺点与模型改进

7.1 模型优点

1. 使用 3 自由度分析浮子的受力状态和运动情况，使用 2 自由度分析振子的受力状态和运动情况，能够简化计算过程，较为容易地得到各绝对量以及振子浮子之间运动的做功情况。
2. 考虑振子和浮子质量的空间分布，求出了振子和浮子转动惯量的精确值，而非使用质点描述其受力情况，更符合实际。

7.2 模型缺点

1. 模型所描述系统受力情况比较复杂，计算过程不够简化。

参考文献

[1] 郑雄波. 两类点吸式波能装置水动力特性研究 [D]. 哈尔滨工程大学,2016.

[2] 王俊杰. 一类波能系统振动机理及其优化控制算法研究 [D]. 哈尔滨工程大学,2020.

[3] 冯建强, 孙诗一. 四阶龙格—库塔法的原理及其应用 [J]. 数学学习与研究,2017(17):3-5.

[4] 吴金明, 陈妮, 钱晨. 惯性式波浪能供电浮标的液压能量转换系统设计研究 [J]. 机械工程学报,2022,58(04):222-231.

[5] 宋玉生, 刘光宇, 朱凌, 王坚. 改进的灰狼优化算法在 SVM 参数优化中的应用 [J]. 传感器与微系统,2022,41(09):151-155.

[6] 郑雄波, 荆丰梅, 何邦琦, 周双红. 基于波浪能的浮式装备随体供电技术研究 [J]. 数字海洋与水下攻防,2020,3(03):236-241.

[7] 元菲. 海蘑菇波能转换系统耦合运动及性能研究 [D]. 哈尔滨工程大学,2019.

[8] Yu T, Tang Y, Shi H, Huang S. Numerical modelling of wave run-up heights and loads on heaving buoy wave energy converter under the influence of regularwaves.Ocean Engergy,2021,225:108670

[9] Cai Y, Huo Y, Shi X, Liu Y.Numerical and experimental research on a resonance-based wave energy converter. Energy Conversion and Management,2022,269:116152

[10] Jiang Nan, Liu Cong, Zhang Xiao, Xu Mingqi. Design and research of wave energy capture device [J]. Chinese Journal of Solar Energy, 2022, 43(08): 447-451.

附录 A 程序代码

A.1 问题一代码

```
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt

plt.rc("font", size=16)

class T1_problem():

    def __init__(self, t_span: tuple = (0, 100),
                 y0: list = [0, 0.5-2433*9.8/80000, 0, 0],
                 f=6250, chuidang_add_mass: float = 1335.535, w: float = 1.4005,
                 chuidang_xingbo=656.3616) -> None:

        self.t_span = t_span
        self.y0 = y0
        self.chuidang_xingbo = chuidang_xingbo
        self.f = f
        self.chuidang_add_mass = chuidang_add_mass
        self.w = w

    def my_ode45(self, t: np.ndarray, z,
                 k: float, a: float):
        '''
        xu'',xu',xd'',xd'
        yu',xu',yd',xd'
        '''

        yu, xu, yd, xd = z#v振,x振,v浮,x浮
        if a == 0:
            return [
                (-80000*(xu-xd-0.5)-k*(yu-yd))/2433-9.8,
                yu,
                (80000*(xu-xd-0.5) +
                 k*(yu-yd) -
                    self.chuidang_xingbo*yd +
                 1025*np.pi*(4/15+2.000010269-xd)*9.8 -
                 9.8*4866 +
                    (self.f*np.cos(self.w*t)))/(4866+self.chuidang_add_mass),
                yd
            ]

        else:
```

```
        return [
            (-80000*(xu-xd-0.5)-(k*((np.abs(yu-yd))**a))*(yu-yd))/2433-9.8,
            yu,
            (80000*(xu-xd-0.5) +
             (k*((np.abs(yu-yd))**a))*(yu-yd) -
                self.chuidang_xingbo*yd +
             1025*np.pi*(4/15+2.000010269-xd)*9.8 -
             9.8*4866 +
                (self.f*np.cos(self.w*t)))/(4866+self.chuidang_add_mass),
            yd
        ]

    def run(self, k: float = 10000, a: float = 0):
        self.sol = solve_ivp(self.my_ode45, t_span=self.t_span, y0=self.y0,
                             args=(k, a),
                             dense_output=True,
                             method='RK45')
        self.t = np.linspace(self.t_span[0], self.t_span[1],
                             (self.t_span[1]-self.t_span[0])*100+1, endpoint=True)
        self.z = self.sol.sol(self.t)
        if a == 0:
            self.P = k*(self.z[0, :]-self.z[2, :])**2
            self.P_average = (0.005*((self.P[:-1]).sum() +
                                     (self.P[1:]).sum()))/(self.t_span[1]-self.t_span[0])
        else:
            self.P = k*(self.z[0, :]-self.z[2, :])**2 * \
                (np.abs((self.z[0, :]-self.z[2, :])))**a
            self.P_average = (0.005*((self.P[:-1]).sum() +
                                     (self.P[1:]).sum()))/(self.t_span[1]-self.t_span[0])

    def show_power(self):
        plt.figure(figsize=(12, 9))

        plt.plot(self.t, self.P)
        plt.xlabel('t/s')
        plt.ylabel('P/(J/s)')
        plt.legend(['功率'], shadow=True, prop={"family": "SimSun", "size": 16})
        plt.show()

    def show_x(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[1, 3], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel("x/m")
        plt.show()
```

```
    def show_v(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[0, 2], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('v/(m/s)')
        plt.show()

    def show_x_interval(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, self.z[1, :]-self.z[3,:])
        plt.legend(['位移差'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('x/m')
        plt.show()
```

```
from T1_problem import T1_problem
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rc("font", size=16)


prob1 = T1_problem(t_span=(0, 200))

prob1.run(k=10000, a=0)
#
# print(prob1.P_average)
prob1.z[1, :] -= prob1.z[1, 0]
prob1.z[1, [1000, 2000, 4000, 6000, 10000]] # 振子
prob1.z[3, [1000, 2000, 4000, 6000, 10000]] # 浮子

prob1.z[0, [1000, 2000, 4000, 6000, 10000]] # 振子速度
prob1.z[2, [1000, 2000, 4000, 6000, 10000]] # 浮子速度

prob1.show_x()
prob1.show_x_interval()

t_tmp = prob1.t
x_tmp1 = prob1.z[[1, 3], :]
```

```
v_tmp1 = prob1.z[[0, 2], :]

prob1.run(k=10000, a=0.5)

print(prob1.P_average)

prob1.z[1, :] -= prob1.z[1, 0]
prob1.z[1, [1000, 2000, 4000, 6000, 10000]] # 振子
prob1.z[3, [1000, 2000, 4000, 6000, 10000]] # 浮子

prob1.z[0, [1000, 2000, 4000, 6000, 10000]] # 振子速度
prob1.z[2, [1000, 2000, 4000, 6000, 10000]] # 浮子速度

x_tmp2 = prob1.z[[1, 3], :]
v_tmp2 = prob1.z[[0, 2], :]

fig = plt.figure(figsize=(12, 9), dpi=120)
plt.subplot(2, 1, 1)
plt.title("线性阻尼系数", fontdict=dict(family="SimSun"))
plt.plot(t_tmp, x_tmp1.T)
# plt.xlabel('t/s')
plt.ylabel('x/m')
plt.legend(['浮子', '振子'], shadow=True, prop={
    "family": "SimSun", "size": 16})

plt.subplot(2, 1, 2)
plt.title("幂次阻尼系数", fontdict=dict(family="SimSun"))
plt.plot(t_tmp, x_tmp2.T)
plt.xlabel('t/s')
plt.ylabel('x/m')
plt.legend(['浮子', '振子'], shadow=True, prop={
    "family": "SimSun", "size": 16})
plt.show()


fig = plt.figure(figsize=(12, 9), dpi=120)
plt.subplot(2, 1, 1)
plt.title("线性阻尼系数", fontdict=dict(family="SimSun"))
plt.plot(t_tmp, v_tmp1.T)
# plt.xlabel('t/s')
plt.ylabel('v/(m/s)')
plt.legend(['浮子', '振子'], shadow=True, prop={
    "family": "SimSun", "size": 16})

plt.subplot(2, 1, 2)
plt.title("幂次阻尼系数", fontdict=dict(family="SimSun"))
plt.plot(t_tmp, v_tmp2.T)
```

```
plt.xlabel('t/s')
plt.ylabel('v/(m/s)')
plt.legend(['浮子', '振子'], shadow=True, prop={
    "family": "SimSun", "size": 16})
plt.show()

t_tmp[:17941:20]
x_tmp1[0, :17941:20] # 浮子
x_tmp1[1, :17941:20] # 振子
v_tmp1[0, :17941:20] # 浮子
v_tmp1[1, :17941:20] # 振子

dt=np.vstack((t_tmp[:17941:20],x_tmp1[1, :17941:20],v_tmp1[1, :17941:20],x_tmp1[0,
    :17941:20],v_tmp1[0, :17941:20])).T
df=pd.DataFrame(dt)

df.to_excel("t1-1.xlsx")

t_tmp[:17941:20]
x_tmp2[0, :17941:20] # 浮子
x_tmp2[1, :17941:20] # 振子
v_tmp2[0, :17941:20] # 浮子
v_tmp2[1, :17941:20] # 振子

dt=np.vstack((t_tmp[:17941:20],x_tmp2[1, :17941:20],v_tmp2[1, :17941:20],x_tmp2[0,
    :17941:20],v_tmp2[0, :17941:20])).T
df=pd.DataFrame(dt)

df.to_excel("t1-2.xlsx")
```

A.2 问题二代码

```
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from T1_problem import T1_problem
from GWO import GWO

plt.rc("font", size=12)
np.random.seed(42)

prob2 = T1_problem(w=2.2143, chuidang_add_mass=1165.992,
                chuidang_xingbo=167.8395, f=4890,t_span=(0,100))
```

```
class optimize(GWO):

    def __init__(self,size: int = 100, iter_num=200, lb=-10, ub=10) -> None:
        super().__init__(dim=1, size=size, iter_num=iter_num, lb=lb, ub=ub)

    def apply_along(self, arr) -> float:
        prob2.run(k=arr[0]*5000+50000,
                  # a=0.5+arr[1]*0.05
                  a=0
                  )

        return -(prob2.P_average)

class optimize_2(GWO):

    def __init__(self,size: int = 100, iter_num=200, lb=-10, ub=10) -> None:
        super().__init__(dim=2, size=size, iter_num=iter_num, lb=lb, ub=ub)

    def apply_along(self, arr) -> float:
        prob2.run(k=arr[0]*5000+50000,
                  a=0.5+arr[1]*0.05

                  )

        return -(prob2.P_average)


#智能 单变量
sols = optimize(size=12,iter_num=150)
sols.run(show_fitness=True)
k_1=sols.gbest[0]*5000+50000
best1=-sols.g_fitness
print(k_1)
prob2.run(k=k_1)

#网格 并画最优点
l=[]
for i in range(1001):
    prob2.run(k=100*i,a=0)
    l.append(prob2.P_average)

plt.figure(figsize=(12,9))
plt.plot(l)
plt.xlabel("阻尼系数 N*s/m",fontdict=dict(family="SimSun"))
plt.xticks([0,200,400,600,800,1000],[0,20000,40000,60000,80000,100000])
plt.ylabel("发电功率/W",fontdict=dict(family="SimSun"))
```

```
plt.scatter(k_1/100,best1,c='r',s=100)
plt.show()


#智能 双变量
sols = optimize_2(size=36,iter_num=50)
sols.run(show_fitness=True)
k_2=sols.gbest[0]*5000+50000
a_2=sols.gbest[1]*0.05+0.5
best2=-sols.g_fitness
print(k_2)
print(a_2)
prob2.run(k=k_2,a=a_2)

l1=[]
for i in range(101):
    for j in range(101):
        prob2.run(k=1000*i,a=0.01*j)
        l1.append(prob2.P_average)

l1=np.array(l1).reshape((101,101))

x_space=np.linspace(0,101,101,endpoint=False)
X,Y=np.meshgrid(x_space,x_space)

fig=plt.figure(figsize=(12,9),dpi=120)
ax = Axes3D(fig)
ax.plot_surface(X,Y,l1[:,::-1],cmap=plt.cm.hot_r)
plt.xlabel("幂指数a",fontdict=dict(family="SimSun",size=16))
plt.ylabel('阻尼系数 N*s/m',fontdict=dict(family="SimSun",size=16))
ax.set_zlabel("发电功率/W",fontdict=dict(family="SimSun",size=16))
plt.xticks([0,20,40,60,80,100],[1,0.8,0.6,0.4,0.2,0])
plt.yticks([0,20,40,60,80,100],[0,20000,40000,60000,80000,100000])
plt.show()

#固定最优k 找a
l2=[]
for i in range(101):
    prob2.run(k=k_1,a=0.01*i)
    l2.append(prob2.P_average)

plt.figure(figsize=(12,9))
plt.plot(l2)
plt.xlabel("幂指数a",fontdict=dict(family="SimSun"))
plt.xticks([0,20,40,60,80,100],[0,0.2,0.4,0.6,0.8,1])
plt.ylabel("发电功率/W",fontdict=dict(family="SimSun"))
plt.show()
```

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

np.random.seed(42)


class GWO:
    '''
    GWO: minimize fn(apply_along)
    apply_along: 目标函数
    dim: 空间维度
    size: 粒子规模
    x_range: 自变量上下界
    a=2: 收敛系数
    iter_num: 迭代次数
    ...
    X: 自变量
    p_fitness: 集体适应度
    g_fitness: 最佳适应度
    fitness_value_list: 每次迭代最佳适应度, 作图用
    '''

    def __init__(self, dim: int, size: int = 100,
                 iter_num=200, lb=-10, ub=10) -> None:
        '''
        apply_along: 目标函数
        dim: 空间维度
        size: 粒子规模
        x_range: 自变量上下界
        w,c1,c2: 速度函数的系数
        iter_num: 迭代次数
        '''

        self.dim = dim
        self.size = size
        self.a = 2
        self.iter_num = iter_num

        self.lb = lb
        self.ub = ub

        self.fitness_value_list = []

        self.X = np.random.uniform(
```

```
            self.lb, self.ub, size=(self.dim, self.size))
        # self.X=np.zeros(shape=(self.dim,self.size))
        # for i in range(self.dim):
        #    self.X[i,:]=np.linspace(self.lb,self.ub,self.size,endpoint=True)

    def show_fitness(self):

        figfit = plt.figure(figsize=(12, 9))
        ax = figfit.add_subplot(1, 1, 1)
        ax.plot(self.fitness_value_list)
        plt.xlabel("迭代次数", fontdict=dict(family="SimSun"), size=16)
        plt.ylabel("目标函数值", fontdict=dict(family="SimSun"), size=16)
        figfit.show()

    def fitness_func(self) -> np.ndarray:
        '''
        适应度函数，返回shape=(1, size)的float数组
        '''

        ret = np.zeros(self.size, dtype=int)
        ret = np.apply_along_axis(self.apply_along, 0, self.X)
        return ret

    def apply_along(self, arr: np.ndarray) -> float:
        '''
        apply_along函数，应该在继承中被覆盖
        '''

        return 0

    def run(self, show_fitness=False) -> np.ndarray:
        self.p_fitness = self.fitness_func()
        self.g_fitness = self.p_fitness.min()
        self.fitness_value_list.append(self.g_fitness)
        #self.pbest = self.X.copy()
        self.gbest = self.X[:, self.p_fitness.argmin()].copy()
        # print(self.X)
        if show_fitness == True:
            print(self.fitness_value_list[-1])

        for i in range(1, self.iter_num):
            self.a = 2-2*i/self.iter_num
            self.X = self.position_update()

            self.p_fitness = self.fitness_func()
            self.g_fitness = self.p_fitness.min()
```

```
            self.fitness_value_list.append(self.g_fitness)
            #self.pbest = self.X.copy()
            self.gbest = self.X[:, self.p_fitness.argmin()].copy()
            # print(self.X)
            if show_fitness == True:
                print(self.fitness_value_list[-1])

        print("最优值是: %.5f" % self.fitness_value_list[-1])
        print("最优解是: ", self.gbest)
        self.show_fitness()
        return self.gbest

    def position_update(self) -> np.ndarray:
        alpha, beta, delta = self.p_fitness.argsort()[:3]
        #r1 = np.random.uniform(-1,1,size=(self.dim,1))
        #r2 = np.random.uniform(-1,1,size=(self.dim,1))

        C = np.random.uniform(0, 2, size=(self.dim, 3))
        A = np.random.uniform(-self.a, self.a, size=(self.dim, 3))

        D = self.X[:, [alpha, beta, delta]]*C

        D1 = np.abs(self.X-D[:, [0]])
        D1[:, alpha] = 0
        X1 = self.X-A[:, [0]]*D1

        D2 = np.abs(self.X-D[:, [1]])
        D2[:, [alpha, beta]] = 0
        X2 = self.X-A[:, [1]]*D2

        D3 = np.abs(self.X-D[:, [2]])
        D3[:, [alpha, beta, delta]] = 0
        X3 = self.X-A[:, [2]]*D3

        m_X = (X1+X2+X3)/3

        m_X[m_X > self.ub] = self.ub
        m_X[m_X < self.lb] = self.lb
        return m_X
```

A.3 问题三代码

```
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt
```

```
plt.rc("font", size=16)


class T3_problem():
    J=(7+np.sqrt(1.64))*4866*(30+81/4+2*np.sqrt(1.64)*(1/8+4/25+16/75))

    def __init__(self, t_span: tuple = (0, 100),
                 y0: list = [0,0,0,0,0, 0.5-2433*9.8/80000, 0, 0],
                 f=6250,
                 chuidang_add_mass: float = 1335.535,
                 w: float = 1.4005,
                 chuidang_xingbo: float = 656.3616,
                 L=1690,
                 zongyao_xingbo: float = 654.3383,
                 zongyao_add_mass: float = 7001.914,
                 # jingshui_huifu=8890.7
                 ) -> None:

        self.t_span = t_span
        self.y0 = y0
        self.chuidang_xingbo = chuidang_xingbo
        self.f = f
        self.chuidang_add_mass = chuidang_add_mass
        self.w = w
        self.zongyao_add_mass = zongyao_add_mass
        self.zongyao_xingbo = zongyao_xingbo
        self.L = L

    def my_ode45(self, t: np.ndarray, z,
                 k_line: float, k_circle: float):
        '''
        r'', r', theta_v'', theta_v', y'', y', x'', x', theta_f'', theta_f'
        r1', r', theta_v1', theta_v', y1', y', x1', x', theta_f1', theta_f'
        '''

        theta2d, theta2, theta1d, theta1, yu, xu, yd, xd = z

        return
            [(250000*(theta1-theta2)+k_circle*(theta1d-theta2d))/(8*2433)/(1/128+((xu-xd+0.5)**3-(xu-xd)**3)/12)
              theta2d,
              (-250000*(theta1-theta2)-k_circle*(theta1d-theta2d)-8890.7*theta1-self.zongyao_xingbo*theta1d+sel
              theta1d,
              (-80000*(xu-xd-0.5)-k_line*(yu-yd))/2433-9.8,
              yu,
              (80000*(xu-xd-0.5) +
               k_line*(yu-yd) -
```

```
                self.chuidang_xingbo*yd +
            1025*np.pi*(4/15+2.000010269-xd)*9.8 -
            9.8*4866 +
                (self.f*np.cos(self.w*t)))/(4866+self.chuidang_add_mass),
            yd
        ]

    def run(self, k_line=10000, k_circle=1000):
        self.sol = solve_ivp(self.my_ode45, t_span=self.t_span, y0=self.y0,
                             args=(k_line, k_circle),
                             dense_output=True,
                             method='RK45')
        self.t = np.linspace(self.t_span[0], self.t_span[1],
                             (self.t_span[1]-self.t_span[0])*100+1, endpoint=True)
        self.z = self.sol.sol(self.t)
        self.P = k_line*(self.z[4, :]-self.z[6,
            :])**2+k_circle*(self.z[0,:]-self.z[2,:])*(self.z[1,:]-self.z[3,:])
        self.P_average = (0.005*((self.P[:-1]).sum() +
                                 (self.P[1:]).sum()))/(self.t_span[1]-self.t_span[0])

    def show_power(self):
        plt.figure(figsize=(12, 9))

        plt.plot(self.t, self.P)
        plt.xlabel('t/s')
        plt.ylabel('P/(J/s)')
        plt.legend(['功率'], shadow=True, prop={"family": "SimSun", "size": 16})
        plt.show()

    def show_theta(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[1, 3], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel("x/m")
        plt.show()

    def show_omega(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[0, 2], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('v/(m/s)')
        plt.show()
```

```
    def show_theta_interval(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, self.z[1, :]-self.z[3,:])
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('v/(m/s)')
        plt.show()

    def show_x(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[5, 7], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel("x/m")
        plt.show()

    def show_v(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, (self.z[[4, 6], :]).T)
        plt.legend(['振子', '浮子'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('v/(m/s)')
        plt.show()

    def show_x_interval(self):
        plt.figure(figsize=(12, 9), dpi=120)
        plt.plot(self.t, self.z[5, :]-self.z[7,:])
        plt.legend(['位移差'], shadow=True, prop={
                  "family": "SimSun", "size": 16})
        plt.xlabel('t/s')
        plt.ylabel('x/m')
        plt.show()
```

```
from T3_problem import T3_problem
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rc("font", size=16)
```

```
prob1 = T3_problem(t_span=(0, 200), w=1.7152,
                chuidang_add_mass=1028.876, zongyao_add_mass=7001.914,
                chuidang_xingbo=683.4558, zongyao_xingbo=654.3383,
                f=3640, L=1690
                )

prob1.run()
prob1.show_theta()

t = prob1.t.reshape(-1, 1)[:7321:20]

theta_fuzi = prob1.z[3, :].reshape(-1, 1)[:7321:20]
omega_fuzi = prob1.z[2, :].reshape(-1, 1)[:7321:20]
theta_zhenzi = prob1.z[1, :].reshape(-1, 1)[:7321:20]
omega_zhenzi = prob1.z[0, :].reshape(-1, 1)[:7321:20]

v_zhenzi = prob1.z[4, :].reshape(-1, 1)[:7321:20]
x_zhenzi = prob1.z[5, :].reshape(-1, 1)[:7321:20]
v_fuzi = prob1.z[6, :].reshape(-1, 1)[:7321:20]
x_fuzi = prob1.z[7, :].reshape(-1, 1)[:7321:20]

dt=np.hstack((t, x_fuzi, v_fuzi, theta_fuzi,
                omega_fuzi, x_zhenzi, v_zhenzi,
                theta_zhenzi, omega_zhenzi))

df = pd.DataFrame(dt)

df.to_excel("3.xlsx")
```

A.4 问题四代码

```
from T3_problem import T3_problem
import numpy as np
from scipy.integrate import solve_ivp
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

plt.rc("font", size=16)

from GWO import GWO

np.random.seed(42)

prob1 = T3_problem(t_span=(0, 100),w=1.9806,
chuidang_add_mass=1091.099,zongyao_add_mass=7142.493,
```

```
chuidang_xingbo=528.5018,zongyao_xingbo=1655.909,
f=1760,L=2140
)

class optimize_2(GWO):

    def __init__(self,size: int = 100, iter_num=200, lb=-10, ub=10) -> None:
        super().__init__(dim=2, size=size, iter_num=iter_num, lb=lb, ub=ub)

    def apply_along(self, arr) -> float:
        prob1.run(k_line=arr[0]*5000+50000,
                  k_circle=50000+arr[1]*5000
                  )

        return -(prob1.P_average)

sols = optimize_2(size=7,iter_num=50)
sols.run(show_fitness=True)
k_line=sols.gbest[0]*5000+50000
k_circle=sols.gbest[1]*5000+50000
best2=-sols.g_fitness
print(k_line)
print(k_circle)
prob1.run(k_line=k_line,k_circle=k_circle)

prob1.show_theta()
```