# 基于差分法和模拟退火法的波浪能最大输出功率设计


## 摘要

本文通过建立**微分方程模型**用**差分法**研究波浪能装置的运动过程，并在此基础上使用**模拟退火法**求出阻尼器在不同条件下使得波浪能装置**输出功率最大**的**阻尼系数**。[1]

对于问题一，先用牛顿第二定律和速度的定义确定**微分方程模型**，再通过使用**差分法**和对初值条件的**迭代**分别计算出阻尼器的阻尼系数为常值 $10000N \cdot s \cdot m^{-1}$ 时和阻尼系数与浮子和振子的相对速度的绝对值的幂成正比时浮子和振子在波浪激励力作用下在前 40 个波浪周期的垂荡位移和速度。

对与问题二，先随机产生一个变量值，然后用**模拟退火**的方法在一定温度下不断产生新解或保留解然后逐渐降温得到稳定的最优解：第一小问的**最大输出功率**为 $280W$，**最优阻尼系数**为 $37200N \cdot s \cdot m^{-1}$，第二小问的的**最大输出功率**为 $280W$，而**最优阻尼系数**的比例系数和幂系数的组合并不唯一，其中一种组合是比例系数为 42796，幂系数为 0.063146。

对于问题三，利用**微分方程模型**将垂荡和纵摇运动放在极小的时间尺度内研究，从而可以以较小的误差假设垂荡和纵摇互不影响实现运算简化。然后利用牛顿第二运动定律和差分法将两个运动分别求解，得到浮子和振子在各个时刻的垂荡位移与速度和纵摇角位移与角速度。

对于问题四，在假设垂荡和纵摇互不影响的前提下，可以采用与问题二相同的**模拟退火法**求解，通过观察计算机的多次运算结果发现最大输出功率为 $492W$，两个阻尼器的阻尼系数搭配并不唯一，其中一种搭配是直线阻尼器的阻尼系数为 20420，旋转阻尼器的**阻尼系数**为 84657。



**关键词：** 微分方程模型　差分法　模拟退火法　阻尼系数　最大输出功率

# 1 问题重述

## 1.1 问题背景

随着全球经济的发展，人类对于能源的需求量不断增大，由化石能源的过度使用产生的温室效应对人类自身的威胁逐渐显现时，清洁的可再生能源就自然而然的进入人类视线。波浪能就是一种绿色清洁的并且取之不尽，用之不竭的可再生能源，如果能有效利用这种能源，那将是人类的福音，而能否高效率的利用波浪能便是能否大面积推广的关键因素之一。

## 1.2 问题提出

波浪能装置是由浮子，振子、中轴和能量输出系统（包括弹簧和阻尼器）组成的装置，它能够将波浪能俘获转化成为机械能进而转化为人类可用的电能。波浪装置的参数在附件 4 给出，请建立数学模型解决以下问题：

**问题**一：假设浮子只做垂荡运动，建立浮子和振子的运动模型，分别计算当直线阻尼器的阻尼系数为 10000 $\mathrm{N\cdot s/m}$ 和直线阻尼器的阻尼系数与浮子和振子的相对速度的绝对值的 0.5 次幂成正比，其中比例系数取 10000 时浮子和振子在波浪激励力作用下前 40 个波浪周期内时间间隔为 0.2 s 的垂荡位移和速度并将计算结果分别存放在表格 result1-1.xlsx 和表格 result1-2.xlsx 中。再对两种情况分别列出 10 s、20 s、40 s、60 s、100 s 时浮子与振子的垂荡位移和速度。

**问题**二：假设浮子只做垂荡运动，利用附件 3 和附件 4 中的参数值分别建立阻尼系数为 0 到 100000 的某个常量时和阻尼系数与浮子和振子的相对速度的绝对值的幂成正比，比例系数是 0 到 100000 的某个值，幂指数是 0 到 1 内某个值时直线阻尼器的最优阻尼系数的数学模型使得能量输出系统系统的平均输出功率最大并分别求出两种情况下的最大输出功率和相应的阻尼系数。

**问题**三：假设浮子只做垂荡和纵摇运动，利用附件 3 和附件 4 中的参数值建立浮子与振子的运动模型计算浮子与振子在波浪激励力和波浪激励力矩作用下前 40 个波浪周

期内时间间隔为 0.2 s 的垂荡位移与速度和纵摇角位移与角速度并将结果存放在表格 result3.xlsx 中同时在论文中分别给出 10 s、20 s、40 s、60 s、100 s 时刻，浮子与振子的垂荡位移与速度和纵摇角位移与角速度。其中直线阻尼器的阻尼系数为 10000 N · s/m，旋转阻尼器的阻尼系数为 1000 N · m · s。

**问题四**：假设浮子在波浪中只做垂荡和纵摇运动，利用附件 3 和附件 4 中的参数值建立确定直线阻尼器和旋转阻尼器最优阻尼系数的数学模型计算当直线阻尼器和旋转阻尼器的阻尼系数都在 0 到 100000 范围取值时的最大输出功率并给出与此对应的两个最优阻尼系数。

# 2 问题分析

## 2.1 问题一的分析

问题一需要求出在给定直线阻尼器阻尼系数和波浪激励力的情况下计算浮子与振子在某个时刻的垂荡位移和速度。由于模型涉及瞬时速度和变力，自然而然用到牛顿第二定律和微分方程，然后结合初值条件再通过差分法进行各个时刻对力的求解，最后通过差分后的方程进行迭代求出各个时刻的垂荡位移和速度。

## 2.2 问题二的分析

问题二需要求出最优阻尼系数使得输出功率最大，这是一个优化问题。问题二的第一种情况只有一个变量需要求最优解，可以采用变步长模拟退火算法；问题二的第二种情况有两个变量需要同时求最优解，可以对双变量随机初始化，然后分别对每一个变量使用模拟退火算法，通过多次运行取最大则可以找到全局最优解，避免结果落入局部最优解。

### 2.3 问题三的分析

问题三需要求出在给定直线阻尼器阻尼系数、旋转阻尼器阻尼系数、波浪激励力和波浪激励力矩的情况下计算浮子与振子在某个时刻的垂荡位移和速度和纵摇角位移与角速度。问题三与问题一相比，浮子的运动增加了纵摇，在问题一的条件基础上增加了波浪激励力矩和旋转阻尼器，与第一问基本相似，只要假设在极短的时间间隔内，垂荡和纵摇两个运动是分别进行的就可以忽略两者的相互影响，从而将问题分解成两个与第一问相似的问题，同样可以使用差分迭代的方法求解出各个时刻浮子与振子的垂荡位移与速度和纵摇角位移与角速度。

### 2.4 问题四的分析

问题四需要求直线阻尼器和旋转阻尼器阻的最优阻尼系数使得输出功率最大。与问题二的第二种情况一样有两个变量需要求最优，故仍然可以采用模拟退火算法算法多次运行找到全局最优解。

## 3 模型假设

1、忽略中轴、底座、隔层及 PTO 的质量和各种摩擦。

2、假设在 0.01s 内，速度、加速度、角速度和角加速度恒定。

3、假设在极短时间内垂荡和纵摇分别独立运动，不会互相影响。

4、假设问题一和问题二中向上为正方向，假设问题三和问题四中浮子的竖直向上为正方向，忽略水平方向的速度和位移，以绕圆锥顶点顺时针方向为正方向，振子以和中轴平行向上的方向为正方向，以绕转轴中心顺时针为正方向。

5、假设问题一和问题二中浮子的锥体部分不会浮出水面。

6、平均功率由前 40 个周期的平均功率决定。

7、纵摇时振子底面到转轴中心点的距离近似等于弹簧的长度。

在一个极小的时间间隔内，速度和加速度的变化是极其有限的，所以将一个计算时

间内的速度、加速度看作恒定的值是合理的; 将短时间内的垂荡和纵摇看作两个独立的运动可以有效避免繁杂的受力分析并得到合理近似的计算结果，所以假设三是合理且必要的.

# 4 符号说明

| 序号 | 符号 | 含义 | 单位 |
|---|---|---|---|
| 1 | $v_1$ | 浮子的速度 | $m \cdot s^{-1}$ |
| 2 | $v_2$ | 振子的速度 | $m \cdot s^{-1}$ |
| 3 | $s_1$ | 浮子的位移 | $m$ |
| 4 | $s_2$ | 振子的位移 | $m$ |
| 5 | $m_1$ | 浮子的质量 | $kg$ |
| 6 | $m_2$ | 振子的质量 | $kg$ |
| 7 | $m_3$ | 垂荡附加质量 | $kg$ |
| 8 | $f_1$ | 静水恢复力 | $N$ |
| 9 | $f_2$ | 弹簧弹力 | $N$ |
| 10 | $f_3$ | 直线阻尼器阻力 | $N$ |
| 11 | $f_4$ | 兴波阻尼力 | $N$ |
| 12 | $f_5$ | 波浪激励力 | $N$ |
| 13 | $f_6$ | 重力偏移中轴产生的偏差力 | $N$ |
| 14 | $f$ | 波浪激励力振幅 | $N$ |
| 15 | $\omega$ | 波浪频率 | $s^{-1}$ |
| 16 | $W_1$ | 直线阻尼器做功 | $J$ |
| 17 | $y_1$ | 直线阻尼器的幂系数 | - |
| 18 | $\omega_1$ | 浮子的角速度 | $s^{-1}$ |

| | | | |
|---|---|---|---|
| 19 | $\omega_2$ | 振子的角速度 | $s^{-1}$ |
| 20 | $\theta_1$ | 浮子的偏转角 | $rad$ |
| 21 | $\theta_2$ | 振子的偏转角 | $rad$ |
| 22 | $I_1$ | 振子的转动惯量 | $kg \cdot m^2$ |
| 23 | $I_2$ | 浮子的转动惯量 | $kg \cdot m^2$ |
| 24 | $I_3$ | 浮子的附加转动惯量 | $kg \cdot m^2$ |
| 25 | $M_1$ | 静水恢复力矩 | $N \cdot m$ |
| 26 | $M_2$ | 扭转弹簧力矩 | $N \cdot m$ |
| 27 | $M_3$ | 旋转阻尼器力矩 | $N \cdot m$ |
| 28 | $M_4$ | 兴波阻尼力矩 | $N \cdot m$ |
| 29 | $M_5$ | 波浪激励力矩 | $N \cdot m$ |
| 30 | $M_6$ | 浮子的重力矩 | $N \cdot m$ |
| 31 | $M_7$ | 振子的重力矩 | $N \cdot m$ |
| 32 | $W_2$ | 旋转阻尼器做功 | $J$ |
| 33 | $f$ | 波浪激励力振幅 | $N$ |
| 34 | $L$ | 波浪激励力矩振幅 | $N \cdot m$ |
| 35 | $a_1$ | 直线阻尼器的阻尼系数 | $N \cdot s \cdot m^{-1}$ |
| 36 | $a_2$ | 旋转阻尼器的阻尼系数 | $N \cdot s \cdot m^{-1}$ |
| 37 | $k_1$ | 垂荡兴波阻尼系数 | $N \cdot s \cdot m^{-1}$ |
| 38 | $k_2$ | 纵摇兴波阻尼系数 | $N \cdot m \cdot s$ |
| 39 | $k_3$ | 静水恢复力矩系数 | $N \cdot m$ |
| 40 | $k_4$ | 扭转弹簧力矩系数 | $N \cdot m$ |
| 41 | $k$ | 弹簧刚度 | $N \cdot m$ |
| 42 | $g$ | 重力加速度 | $m \cdot s^{-2}$ |
| 43 | $\rho$ | 海水的密度 | $kg \cdot m^{-3}$ |

| | | | |
|---|---|---|---|
| 44 | $V_0$ | 初始浮子没入水中的体积 | $m^3$ |
| 45 | $V$ | 实时浮子没入水中的体积 | $m^3$ |
| 46 | $l_0$ | 初始时弹簧的长度 | $m$ |
| 47 | $l$ | 实时弹簧的长度 | $m$ |
| 48 | $x_1$ | 浮子重心离圆锥顶点的距离 | $m$ |
| 49 | $x_2$ | 振子重心离转轴中心点的距离 | $m$ |

# 5 问题一模型的建立与求解

## 5.1 问题一模型的建立

### 5.1.1 问题复述

问题一只考虑波浪能装置垂荡的情况，有两个小问，需要分别求出直线阻尼器的阻尼系数为 $10000N \cdot s \cdot m^{-1}$ 和直线阻尼器的阻尼系数与浮子和振子的相对速度的绝对值的幂成正比，比例系数取 10000，幂指数取 0.5 时浮子和振子在波浪激励力作用下各个时刻的垂荡位移和速度。

### 5.1.2 建模分析

问题一的两个小问都是给出直线阻尼器的阻尼系数求浮子和振子的运动模型，所以两个小问都可以采用微分法来建模，用差分法求解。受力图

### 5.1.3 模型建立

对浮子进行受力分析，由牛顿第二运动定律得：

$$(m_1 + m_3)\frac{dv_1}{dt} = f_1 + f_2 + f_3 + f_4 + f_5$$

对振子进行受力分析，由牛顿第二运动定律得：

$$m_2\frac{dv_2}{dt} = -f_2 - f_3$$

浮子的位移速度关系为

$$\frac{ds_1}{dt} = v_1$$

振子的位移速度关系为

$$\frac{ds_2}{dt} = v_2$$

由题意，差分方程的初值条件为:

$$s_1(0) = 0$$

$$s_2(0) = 0$$

$$v_1(0) = 0$$

$$v_2(0) = 0$$

静水恢复力为

$$f_1 = -\rho g \pi r^2 s_1$$

弹簧弹力

$$f_2 = -k(s_1 - s_2)$$

直线阻尼器阻力

$$f_3 = -a_1(v_1 - v_2)$$

兴波阻尼力

$$f_4 = -k_1 v_1$$

波浪激励力

$$f_5 = f\cos\omega t$$

## 5.2 问题一模型的求解

### 5.2.1 求解与分析

令 $dt = t_n - t_{n-1}$,$dv = v_1(n) - v_1(n-1)$,$t_n$ 表示第 $n$ 个时刻，$v_1(n)$ 表示第 $n$ 个时刻的 $v_1$。

通过已知的初值条件，可由第 n-1 个时刻的速度，位移等已经求出的量，求出第 n 个时刻的第速度位移等未知的量。由牛顿第二运动定律得

$$(m_1+m_3)\frac{v_1(n)-v_1(n-1)}{T}=f_1+f_2+f_3+f_4+f_5$$

$$m_2\frac{v_2(n)-v_2(n-1)}{T}=-f_2-f_3$$

浮子位移与速度关系

$$\frac{s_1(n)-s_1(n-1)}{T}=\frac{v_1(n)+v_1(n-1)}{2}$$

振子位移与速度关系

$$\frac{s_2(n)-s_2(n-1)}{T}=\frac{v_2(n)+v_2(n-1)}{2}$$

### 5.2.2 结果检验

经过运行程序，得到浮子和振子在各个时刻的位移和速度，详细数据见附录。其中浮子和振子在 10s、20s、40s、60s、100s 的位移和速度数据在下表列出，第一问数据见表 2，第二问数据见表 3。

| 时间 (s) | 浮子 | | 振子 | |
|---|---|---|---|---|
| | 位移 (m) | 速度 (m/s) | 位移 (m) | 速度 (m/s) |
| 10 | -0.2654 | -0.6283 | -0.2945 | -0.6825 |
| 20 | -0.6754 | -0.2079 | -0.7288 | -0.2405 |
| 40 | 0.2170 | 0.2800 | 0.2203 | 0.2936 |
| 60 | -0.3903 | -0.4792 | -0.4154 | -0.5191 |
| 100 | -0.1299 | -0.6151 | -0.1349 | -0.6569 |

表 2: 第一问各个时刻浮子和振子的位移和速度

| 时间 (s) | 浮子 | | 振子 | |
|---|---|---|---|---|
| | 位移 (m) | 速度 (m/s) | 位移 (m) | 速度 (m/s) |
| 10 | -0.2654 | -0.6283 | -0.2945 | -0.6825 |
| 20 | -0.6754 | -0.2079 | -0.7288 | -0.2405 |
| 40 | 0.2170 | 0.2800 | 0.2203 | 0.2936 |
| 60 | -0.3903 | -0.4792 | -0.4154 | -0.5191 |
| 100 | -0.1299 | -0.6151 | -0.1349 | -0.6569 |

表 3: 第二问各个时刻浮子和振子的位移和速度

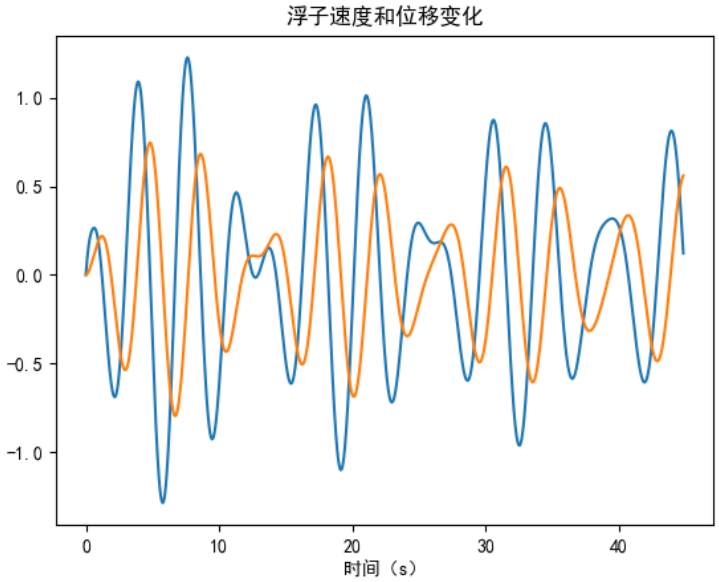


图 1: 第一问浮子速度位移

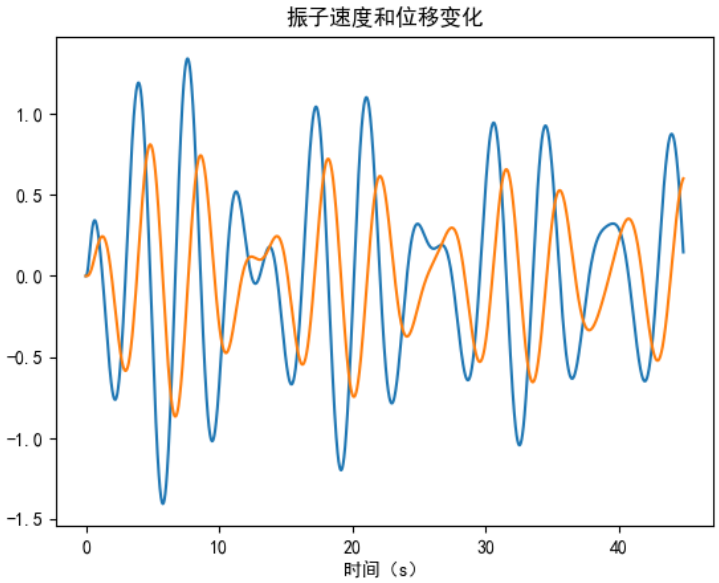


图 2: 第一问振子速度位移

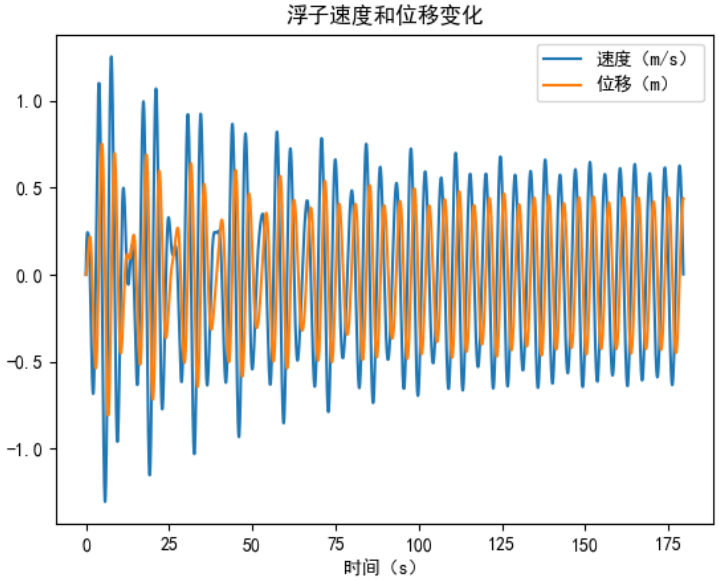


图 3: 第二问浮子速度位移

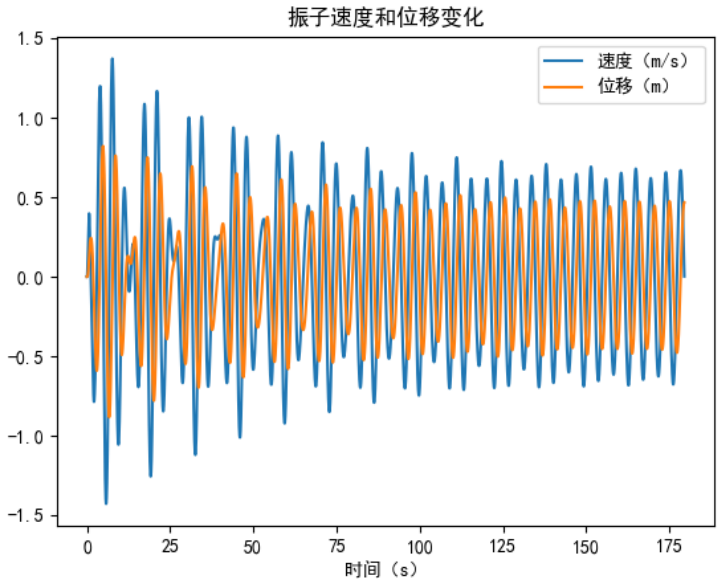


图 4: 第二问振子速度位移

图一图二分别表示第一问浮子和振子的速度位移变化图，图三图四分别表示第二问浮子和振子的速度位移变化图其中蓝色线条表示速度变化，黄色线条表示位移变化。通过观察浮子和振子的速度变化图像可以发现两者的速度与位移都在稳定位置 0 附近上下波动，波动幅度约为一个单位，再结合附件 4 中给出的浮子和振子的质量分析，运行结果符合实际贴近现实是合理的。

# 6 问题二模型的建立与求解

## 6.1 问题二模型的建立

### 6.1.1 问题复述

问题二只考虑波浪能装置垂荡的情况，有两个小问，需要分别求出当直线阻尼器的阻尼系数在区间 $[0, 100000]$ 变动时和阻尼系数与浮子和振子的相对速度的绝对值的幂成正比，比例系数在区间 $[0, 100000]$ 内取值，幂指数在区间 $[0, 1]$ 内取值的情况下使得波浪能装置具有最大输出功率的阻尼系数。

### 6.1.2 建模分析

第一问相当于求一个变量的最大值问题，问题二相当于求解双变量的最大值问题。对于求最值的问题可以使用模拟退火法寻求最优解。模拟退火法基本原理：

模拟退火算法是模拟固体退火过程中固体粒子内能逐渐降低到稳定态的特性在解空间中随机寻找目标函数的全局最优解。在符合条件时则更新原值为迭代后的值，若不符合条件，则以一定概率更新原值从而可以避免得到局部最优解。

### 6.1.3 模型建立

利用问题一的模型：

$$max \quad P = \frac{W}{40}\omega$$

$$W = \sum_{n=1}^{\frac{40}{T_1 W}} W_n$$

$$W_n = f_3(n)|[s_1(n) - s_1(n-1)] - [s_2(n) - s_2(n-1)]|$$

## 6.2 问题二模型的求解

### 6.2.1 算法设计

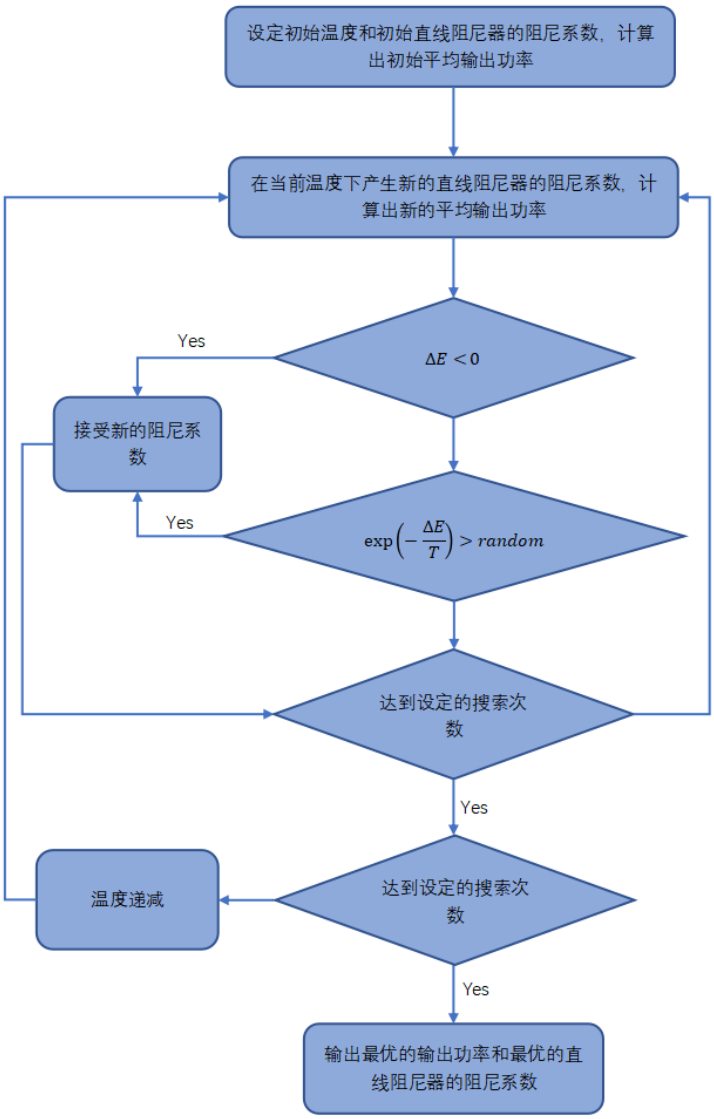


图 5: 第一问模拟退火流程

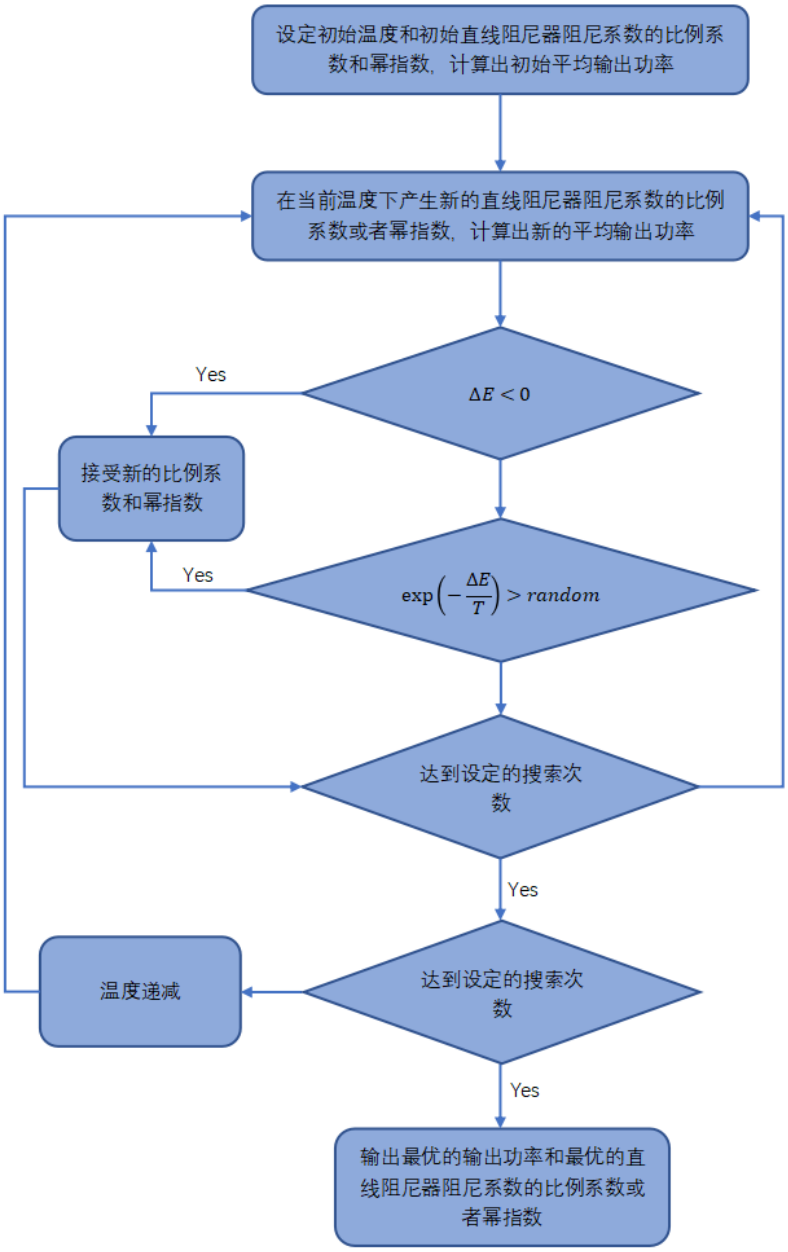


图 6: 第二问模拟退火流程

### 6.2.2 求解与检验

| 模拟退火程序三次运行结果 | | | |
|---|---|---|---|
| | 第一次运行结果 | 第二次运行结果 | 第三次运行结果 |
| 最大输出功率 | 280.3033 | 280.3033 | 280.3032 |
| 最优阻尼系数 | 37198.97 | 37202.50 | 37162.66 |

表 4: 第一问的三次运行结果

| 模拟退火程序三次运行结果 | | | |
|---|---|---|---|
| | 第一次运行结果 | 第二次运行结果 | 第三次运行结果 |
| 最大输出功率 | 280.0271 | 280.1378 | 279.1556 |
| 比例系数 | 42796.13 | 40538.20 | 70308.90 |
| 幂指数 | 0.063146 | 0.027789 | 0.265341 |

表 5: 第二问的三次运行结果

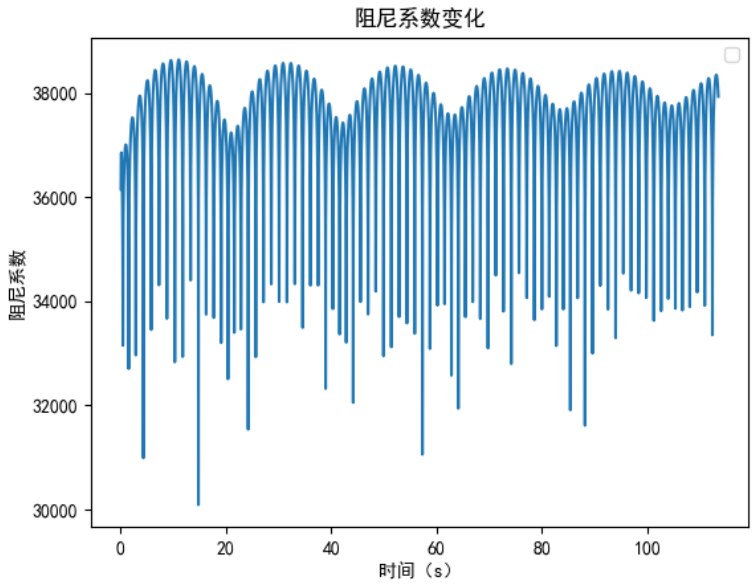


图 7: 第二问最优阻尼系数的变化图

使用模拟退火法运行多次的目的是为了避免随机给定的初值经过迭代后落入局部最优解。

第一问的三次运行结果均显示最大输出功率是 $280W$，最优阻尼系数也稳定在 $37200N \cdot s \cdot m^{-1}$ 附近区间, 由此可以判断出 $280W$ 是最优解。第二问的三次运行结果最大输出功率均在 $280W$ 附近波动，虽然三次运行求得的比列系数和幂指数的配对并不相同，由此可以得出结论，最大输出功率是 $280W$，而达到最大功率时阻尼系数对应的两个变量可以有不同的搭配。

# 7 问题三模型的建立与求解

## 7.1 问题三模型的建立

### 7.1.1 问题复述

问题三考虑浮子既垂荡又纵摇的情形，需要计算浮子与振子在波浪激励力和波浪激励力矩作用下前 40 个波浪周期内时间间隔为 0.2 s 的垂荡位移与速度和纵摇角位移与角速度。

### 7.1.2 建模分析

问题三与问题一相类似，问题三是在问题一的基础上增加了浮子的纵摇，但是在假设的前提下，在极短时间内垂荡和纵摇分别独立运动，不会互相影响，由此便可以采用与第一问同样的微分方程模型，用差分法对两种运动分别迭代求解。

### 7.1.3 模型建立

由角动量定理得

$$(I_1 + I_3)\frac{d\omega_1}{dt} = M_1 + M_2 + M_3 + M_4 + M_5 + M_6$$

$$I_2\frac{d\omega_2}{dt} = -M_2 - M_3 + M_7$$

由牛顿第二运动定律得

$$(m_1 + m_2)\frac{dv_1}{dt} = f_1 + f_4 + f_5 + \cos\theta_2(f_2 + f_3)$$

$$m_2\frac{dv_2}{dt} = -f_2 - f_3 + f_6$$

角速度和角度关系

$$\omega_1 = \frac{d\theta_1}{dt}$$

$$\omega_2 = \frac{d\theta_2}{dt}$$

速度和位移关系

$$v_1 = \frac{ds_1}{dt}$$

$$v_2 = \frac{ds_2}{dt}$$

差分方程的初值条件为

$$v_1 = 0$$

$$v_2 = 0$$

$$\omega_1 = 0$$

$$\omega_2 = 0$$

$$\theta_1 = 0$$

$$\theta_2 = 0$$

$$s_1 = 0$$

$$s_2 = 0$$

静水恢复力

$$f_1 = \rho g(V - V_0)$$

弹簧弹力

$$f_2 = k(l - l_0)$$

直线阻尼器阻力

$$f_3 = -a_1(v_1 \cos\theta_2) - v_2$$

兴波阻尼力

$$f_4 = -k_1 v_1$$

波浪激励力

$$f_5 = f \cos\omega t$$

重力偏移中轴产生的偏差力

$$f_6 = mg(1 - \cos\theta_2)$$

静水恢复力矩

$$M_1 = -k_3\theta_1$$

扭转弹簧力矩

$$M_2 = -k_4(\theta_1 - \theta_2)$$

旋转阻尼器力矩

$$M_3 = -a_2(\omega_1 - \omega_2)$$

兴波阻尼力矩

$$M_4 = -k_2\omega_1$$

波浪激励力矩

$$M_5 = L\cos\omega t$$

浮子的重力矩

$$M_6 = m_1 g x_1 \sin\theta_1$$

振子的重力矩

$$M_7 = m_2 g x_2 \sin\theta_2$$

由平行轴定理可得

$$I_c = \frac{m_2}{12}(r^2 + 3l^2) = \frac{m_2}{12}$$

振子的转动惯量

$$I_2 = I_c + m_2 x_2^2$$

浮子的单位面积质量

$$\sigma = \frac{m_1}{8.28\pi}$$

浮子的重心

$$x_1 = \frac{2\pi\sigma}{m_1}(\int_0^{0.8} \frac{x^2}{0.8}dx + \int_{0.8}^{3.8} xdx + \int_0^1 3.8xdx) = 2.177$$

浮子的转动惯量[2]

$$I_1 = 2\pi\sigma[\int_0^{0.8}(\frac{x^3}{1.024}+\frac{x^3}{0.8})dx+\int_{0.8}^{3.8}(\frac{1}{2}+x^2)dx+\int_0^1(\frac{1}{2}x^3+14.44x)dx]$$

对于 $l(n)$:

垂荡变化为:

$$-\frac{s_1(n)-s_1(n-1)}{\cos\theta_2(n)+s_2(n)-s_2(n-1)}$$

纵摇变化为:

$$0.8\frac{\cos\theta_1(n)-\cos\theta_2(n)}{\cos\theta_2(n)}$$

对于 $V$:

水面上体积为

$$(3.8-\frac{2.8-s_1(n)}{\cos\theta_1(n)})\pi$$

则水下体积为

$$V=(\frac{49}{15}-3.8+\frac{2.8-s_1(n)}{\cos\theta_1(n)})\pi$$

## 7.2 问题三模型的求解

### 7.2.1 算法设计

通过已知的初值条件，可由第 n-1 个时刻的速度，位移等已经求出的量，求出第 n 个时刻的第速度位移等未知的量。由角动量定理得

$$(I_1+I_3)\frac{\omega_1(n)}{T}=M_1+M_2+M_3+M_4+M_5+M_6$$

$$I_2=\frac{\omega_2(n)-\omega_2(n-1)}{T}=-M_2-M_3+M_7$$

由牛顿第二运动定律得

$$(m_1+m_2)\frac{v_1(n)-v_1(n-1)}{T}=f_1+f_4+f_5+\cos\theta_2(f_2+f_3)$$

$$m_2\frac{v_2(n)-v_2(n-1)}{T}=-f_2-f_3+f_6$$

角速度与角度变化关系

$$\omega_1=\frac{\theta_1(n)-\theta_1(n-1)}{T}$$
$$\omega_2=\frac{\theta_2(n)-\theta_2(n-1)}{T}$$

速度与位移变化关系

$$v_1=\frac{s_1(n)-s_1(n-1)}{T}$$
$$v_2=\frac{s_2(n)-s_2(n-1)}{T}$$

### 7.2.2 求解与检验

运行程序后可以得到浮子和振子在各个时刻的垂直位移、垂荡速度、纵摇角位移和纵摇角速度，具体数据可见附录部分。其中在时刻 10s、20s、40s、60s、100s 时浮子和振子的运动数据分别在表 6 和表 7 给出。

| 时间 (s) | 垂荡位移 (m) | 垂荡速度 (m/s) | 纵摇角位移 | 纵摇角速度 ($s^{-1}$) |
|---|---|---|---|---|
| 10 | -0.529814729 | 0.982552298 | -5.93418E-05 | -1.26488E-05 |
| 20 | -0.714955179 | -0.250571613 | 1.3248E-05 | -9.87522E-05 |
| 40 | 0.37065284 | 0.762913329 | -3.11955E-05 | 8.91552E-05 |
| 60 | -0.325491939 | -0.709658161 | 3.66579E-05 | -7.41146E-05 |
| 100 | -0.054859752 | -0.946268642 | 4.89958E-05 | -3.08683E-05 |

表 6: 浮子数据

| 时间 (s) | 垂荡位移 (m) | 垂荡速度 (m/s) | 纵摇角位移 | 纵摇角速度 ($s^{-1}$) |
|---|---|---|---|---|
| 10 | -0.600661181 | 1.052271336 | -1.61604E-05 | -4.19457E-05 |
| 20 | -0.783863243 | -0.29859212 | 1.79152E-05 | -4.02849E-06 |
| 40 | 0.39399833 | 0.851018265 | -2.98992E-05 | -1.62215E-06 |
| 60 | -0.346935309 | -0.786004323 | 1.34963E-05 | 1.83243E-05 |
| 100 | -0.04783448 | -1.036234636 | 1.3885E-06 | 3.30068E-05 |

表 7: 振子数据

分析数据后发现浮子和振子在各个时刻的垂荡位移、垂荡速度、纵摇角位移、纵摇角速度都在包含 0 的一个合理区间内。

# 8 问题四模型的建立与求解

## 8.1 问题四模型的建立

### 8.1.1 问题复述

问题四考虑波浪能装置既垂荡又纵摇的情况，需要建立确定直线阻尼器和旋转阻尼器最优阻尼系数的数学模型，求出当直线阻尼器和旋转阻尼器的阻尼系数均在区间 $[0, 100000]$ 内取一个常值时的最大输出功率以及相应的最优阻尼系数。

### 8.1.2 建模分析

问题四与问题二的第二问类似需要计算双变量情况下的最大值。在假设条件下，极短时间内垂荡和纵摇分别独立运动，不会互相影响。由此仍然可以采用模拟退火算法求出最优解。[3]

### 8.1.3 模型建立

结合问题三的模型有

$$W_n' = M|[\theta_1(n) - \theta_1(n-1)] - [\theta_2(n) - \theta_2(n-1)]|$$

$$max \quad P' = \sum_{0}^{\frac{40}{\omega_1 T}} W_n' + \sum_{0}^{\frac{40}{\omega_1 T}} W_n$$

## 8.2 问题四模型的求解

### 8.2.1 算法设计

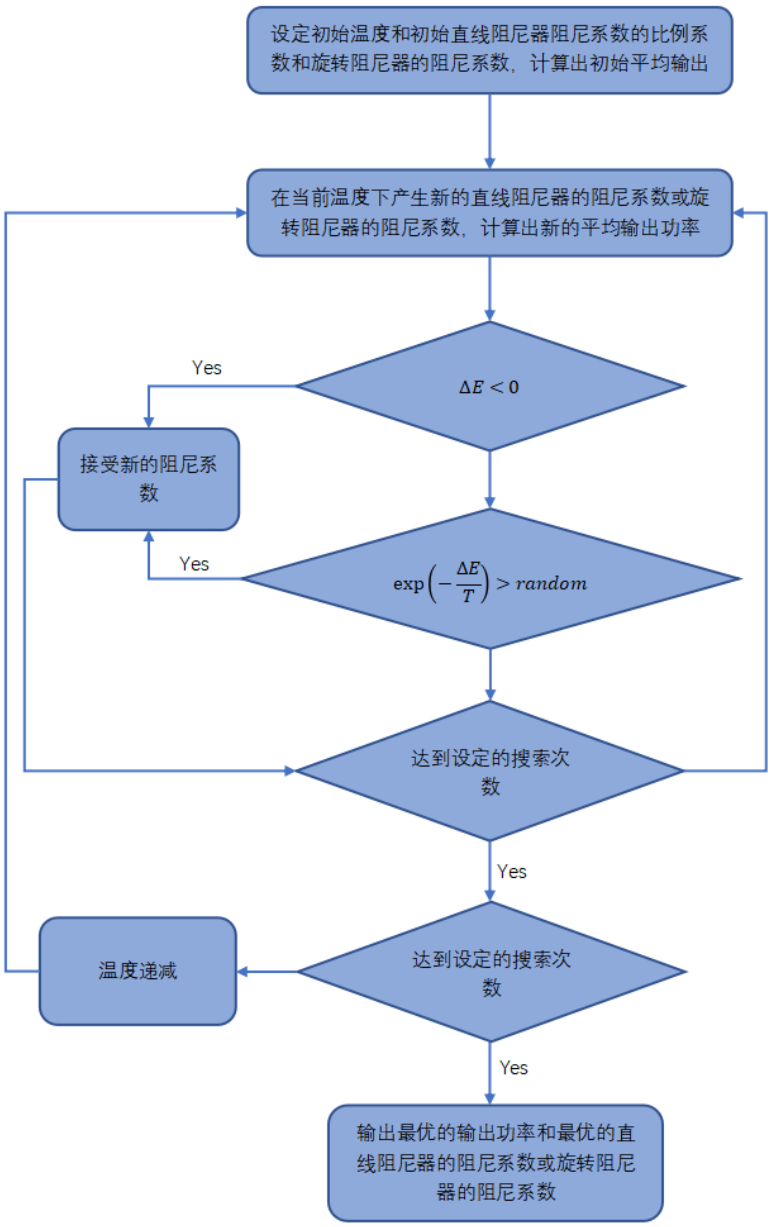


图 8: 问题四模拟退火流程

### 8.2.2 求解与检验

问题四的运行结果为最大功率 $P' = 492W$, 其中一组阻尼系数的组合为直线阻尼器的阻尼系数为 $20420N \cdot s \cdot m^{-1}$，旋转阻尼器的阻尼系数为 $84657N \cdot s \cdot m^{-1}$

经过多次运行，最大功率稳定在 $492W$ 附近，从而可以确定 $492W$ 是全局最优解，而阻尼系数的多种搭配也都符合区间要求，所以结果的合理性得以验证。

# 9 灵敏度检验

对于问题一第一问，当波浪激励力振幅增加 $1000N$ 后，浮子和振子的速度位移数据如表 8 所示。经过对比，

| 时间 (s) | 浮子 | | 振子 | |
|---|---|---|---|---|
| | 位移 (m) | 速度 (m/s) | 位移 (m) | 速度 (m/s) |
| 10 | -0.3079 | -0.7289 | -0.3416 | -0.7917 |
| 20 | -0.7835 | -0.2412 | -0.8454 | -0.2789 |
| 40 | 0.2517 | 0.3249 | 0.2555 | 0.3406 |
| 60 | -0.4527 | -0.5559 | -0.4818 | -0.6022 |
| 100 | -0.1507 | -0.7135 | -0.1565 | -0.7620 |

表 8: 浮子和振子在波浪激励力振幅增加 1000N 后的速度位移图像

对于问题三，当波浪激励力 $1000N$ 波浪激励力矩增加 $300N$ 后，浮子和振子的运动数据分别如表 9 表 10 所示

| 时间 (s) | 垂荡位移 (m) | 垂荡速度 (m/s) | 纵摇角位移 | 纵摇角速度 ($s^{-1}$) |
|---|---|---|---|---|
| 10 | -0.6754 | 1.2525 | -5.9342E-05 | -1.2649E-05 |
| 20 | -0.9114 | -0.3194 | 1.3248E-05 | -9.8752E-05 |
| 40 | 0.4725 | 0.9725 | -3.1195E-05 | 8.9155E-05 |
| 60 | -0.4149 | -0.9046 | 3.6658E-05 | -7.4115E-05 |
| 100 | -0.0699 | -1.2062 | 4.8996E-05 | -3.0868E-05 |

表 9: 浮子的运动数据

| 时间 (s) | 垂荡位移 (m) | 垂荡速度 (m/s) | 纵摇角位移 | 纵摇角速度 ($s^{-1}$) |
|---|---|---|---|---|
| 10 | -0.7657 | 1.3414 | -1.6160E-05 | -4.1946E-05 |
| 20 | -0.9992 | -0.3806 | 1.7915E-05 | -4.0285E-06 |
| 40 | 0.5022 | 1.0848 | -2.9899E-05 | -1.6222E-06 |
| 60 | -0.4422 | -1.0019 | 1.3496E-05 | 1.8324E-05 |
| 100 | -0.0610 | -1.3209 | 1.3885E-06 | 3.3007E-05 |

表 10: 振子的运动数据

# 10 模型的评价与推广

## 10.1 模型的评价

### 10.1.1 模型的优点

1. 模型简化了垂荡和纵摇的关系，将两者分开考虑，在给定基本条件后便可利用计算机的强大计算能力计算出结果。

2. 使用了差分的方法，将两个运动分解成多个极小时间内的运动计算，再结合计算机的强大算力，计算结果的误差就在可以接受的范围内。

3. 该模型具有较高的普适性，对于其他用到弹簧的领域可以类比调用模型。

### 10.1.2 模型的缺点

1. 对于双变量求最优问题使用模拟退火法有一定可能落入局部最优解，需要多次运行程序排除局部最优解。

2. 为了减小计算量，模型将纵摇和垂荡的两个运动分开考虑，这不可避免的产生一定误差。

## 10.2 模型的改进和推广

### 10.2.1 模型的改进

可以通过物理知识建立更精细的微分方程并通过某些数学方法如进行二阶微分，使差分法产生的误差减小。

### 10.2.2 模型的推广

可以将模型推广到减震弹簧的参数设计上使减震效果达到最优。

# 参考文献

[1] 姜楠, 刘聪, 张萧, 徐明奇. 波浪能俘获装置的设计和研究 [J]. 太阳能学报,2022,43(08):447-451.DOI:10.19912/j.0254-0096.tynxb.2020-1349.

[2] 白望望, 杨振斌, 杨德州, 张中丹, 冯智慧. 基于动能定理的发电系统机械转动惯量研究 [J]. 机械研究与应用,2022,35(04):148-152+156.DOI:10.16576/j.ISSN.1007-4414.2022.04.039.

[3] Steinbrunn M ,Moerkotte G, Kemper A. Heuristic and Randomized Optimization for the Join Ordering Problem[J ] . The VLDB Journal , 1997 , 6 (3) :8 - 17.

# 11 附录

问题一第一问 python 程序:

```
import numpy as np                          # 导入模块 numpy 并简写成 np
import pandas as pd                         # 导入模块 pandas 并简写成 pd
import matplotlib.pyplot as plt             # 导入模块 matplotlib.pyplot 并简写成 plt
from math import *                          # 导入模块 math
import random                               # 导入模块 random

####参数
w=1.4005# 入射波浪频率 (s-1)
f=6250# 垂荡激励力振幅 (N)
dt=0.02# 时间步长
T=2*pi/w #周期
m3=1335.535# 垂荡附加质量 (kg)
k1=656.3616# 垂荡兴波阻尼系数 (N·s/m)
m1=4866# 浮子质量 (kg)
r1=1# 浮子底半径 (m)
h11=3# 浮子圆柱部分高度 (m)
h12=0.8# 浮子圆锥部分高度 (m)
m2=2433# 振子质量 (kg)
r2=0.5# 振子半径 (m)
h2=0.5# 振子高度 (m)
```

```
21  rho=1025# 海水的密度 (kg/m3)
22  g=9.8# 重力加速度 (m/s2)
23  F_gangdu1=80000# 弹簧刚度 (N/m)
24  L=0.5# 弹簧原长 (m)
25  S=pi*r1**2# 浮子横截面积
26  zeta=10000# 阻尼系数
27
28  t=np.arange(0,40*T,dt) #时间
29  n=len(t)
30  v1=np.zeros(n, dtype=float)
31  x1=np.zeros(n, dtype=float)
32  v2=np.zeros(n, dtype=float)
33  x2=np.zeros(n, dtype=float)
34  v1[0]=0# 浮子初始位移
35  x1[0]=0# 浮子初始速度
36  v2[0]=0# 振子初始位移
37  x2[0]=0# 振子初始速度
38
39  for i in range(1,n):
40  F_1 = -rho * g * (S * x1[i-1]) # 静水恢复力
41  F_2 = -F_gangdu1 * (x1[i-1] - x2[i-1]) # 弹力
42  F_3 = -zeta * (v1[i-1] - v2[i-1]) # 阻尼器阻力
43  F_4 = -k1 * v1[i-1] # 兴波阻尼力
44  F_5 = f * cos(w * i*dt) # 波浪激励力
45  F_h1 = F_1 + F_2 + F_3 + F_4 + F_5 # 浮子受力
46  F_h2 = -F_2 - F_3 # 振子受力
```

```
47          v1[i]=dt/(m1+m3)*F_h1+v1[i-1]#浮子速度
48          v2[i] = dt / m2 * F_h2 + v2[i - 1]#振子速度
49          x1[i]=0.5*(v1[i]+v1[i-1])*dt+x1[i-1]#浮子位移
50          x2[i]=0.5*(v2[i]+v2[i-1])*dt+x2[i-1]#振子位移
51
52          plt.rcParams['font.sans-serif']=['SimHei']
53          plt.rcParams['axes.unicode_minus']=False
54          plt.figure()
55          plt.plot(t,v2,label='速度 (m/s) ')
56          plt.plot(t,x2,label='位移 (m) ')
57          plt.xlabel('时间 (s) ')
58          plt.title('振子速度和位移变化')
59          plt.legend()
60
61          plt.figure()
62          plt.plot(t,v1,label='速度 (m/s) ')
63          plt.plot(t,x1,label='位移 (m) ')
64          plt.xlabel('时间 (s) ')
65          plt.title('浮子速度和位移变化')
66          plt.legend()
67
68          plt.figure()
69          plt.subplot(2,2,1)
70          plt.plot(t,v1,color='blue',linewidth=1.0)
71          plt.xlabel('时间 (s) ')
72          plt.ylabel('速度 (m/s) ')
```

```
73          plt.title('浮子速度变化')
74
75          plt.subplot(2,2,2)
76          plt.plot(t,x1,color='blue',linewidth=1.0)
77          plt.xlabel('时间 (s) ')
78          plt.ylabel('浮子位移 (m) ')
79          plt.title('浮子位移变化')
80
81          plt.subplot(2,2,3)
82          plt.plot(t,v2,color='blue',linewidth=1.0)
83          plt.xlabel('时间 (s) ')
84          plt.ylabel('速度 (m/s) ')
85          plt.title('振子速度变化')
86
87          plt.subplot(2,2,4)
88          plt.plot(t,x2,color='blue',linewidth=1.0)
89          plt.xlabel('时间 (s) ')
90          plt.ylabel('振子位移变化 (m) ')
91          plt.title('振子位移变化')
92
93          plt.subplots_adjust(left=None,bottom=None,right=None,
                top=None,wspace=0.5,hspace=0.5)
94          plt.show()
```

问题一第二问 python 程序:

```
1                   import numpy as np                  # 导入模块
                        numpy 并简写成 np
```

```
2  import pandas as pd                    # 导入模块
       pandas 并简写成 pd
3  import matplotlib.pyplot as plt        # 导入模块
       matplotlib.pyplot 并简写成 plt
4  from math import *                     # 导入模块
       math
5  import random                          # 导入模块
       random
6
7  ####参数
8  w=1.4005# 入射波浪频率 (s-1)
9  f=6250# 垂荡激励力振幅 (N)
10 dt=0.02# 时间步长
11 T=2*pi/w #周期
12 m3=1335.535# 垂荡附加质量 (kg)
13 k1=656.3616# 垂荡兴波阻尼系数 (N·s/m)
14 m1=4866# 浮子质量 (kg)
15 r1=1# 浮子底半径 (m)
16 h11=3# 浮子圆柱部分高度 (m)
17 h12=0.8# 浮子圆锥部分高度 (m)
18 m2=2433# 振子质量 (kg)
19 r2=0.5# 振子半径 (m)
20 h2=0.5# 振子高度 (m)
21 rho=1025# 海水的密度 (kg/m3)
22 g=9.8# 重力加速度 (m/s2)
23 F_gangdu1=80000# 弹簧刚度 (N/m)
```

```
L=0.5# 弹簧原长 (m)
F_gangdu2=250000# 扭转弹簧刚度 (N·m)
#F_t=F_gangdu1*(L1-L)# 弹簧弹力
theta=atan(1/0.8) #圆锥角度
S=pi*r1**2# 浮子横截面积
V0=(m1+m2)/rho # 静止时的排水体积

t=np.arange(0,40*T,dt) #时间
n=len(t)
v1=np.zeros(n, dtype=float)
x1=np.zeros(n, dtype=float)
v2=np.zeros(n, dtype=float)
x2=np.zeros(n, dtype=float)
v1[0]=0# 浮子初始位移
x1[0]=0# 浮子初始速度
v2[0]=0# 振子初始位移
x2[0]=0# 振子初始速度
for i in range(1,n):
F_1 = -rho * g * (S * x1[i-1]) # 静水恢复力
F_2 = -F_gangdu1 * (x1[i-1] - x2[i-1]) # 弹力
a=abs(v1[i - 1] - v2[i - 1])**0.5
zeta1= 10000 * a # 阻尼器阻力
F_3 = -zeta1 * (v1[i - 1] - v2[i - 1]) # 阻尼
    器阻力
F_4 = -k1 * v1[i-1] # 兴波阻尼力
F_5 = f * cos(w * i*dt) # 波浪激励力
```

```
49                  F_h1 = F_1 + F_2 + F_3 + F_4 + F_5  # 浮子受力
50                  F_h2 = -F_2 - F_3  # 振子受力
51                  v1[i]=dt/(m1+m3)*F_h1+v1[i-1]#浮子速度
52                  v2[i] = dt / m2 * F_h2 + v2[i - 1]#振子速度
53                  x1[i]=0.5*(v1[i]+v1[i-1])*dt+x1[i-1]#浮子位移
54                  x2[i]=0.5*(v2[i]+v2[i-1])*dt+x2[i-1]#振子位移
55
56                  plt.rcParams['font.sans-serif']=['SimHei']
57                  plt.rcParams['axes.unicode_minus']=False
58                  plt.figure()
59                  plt.plot(t,v2,label='速度 (m/s) ')
60                  plt.plot(t,x2,label='位移 (m) ')
61                  plt.xlabel('时间 (s) ')
62                  plt.title('振子速度和位移变化')
63                  plt.legend()
64
65                  plt.figure()
66                  plt.plot(t,v1,label='速度 (m/s) ')
67                  plt.plot(t,x1,label='位移 (m) ')
68                  plt.xlabel('时间 (s) ')
69                  plt.title('浮子速度和位移变化')
70                  plt.legend()
71                  plt.show()
```

问题二第一问 python 程序:

```
1                   from math import *                  # 导入math
                        模块
```

```
2  import random                          # 导入
       random模块
3  import pandas as pd                    # 导入
       pandas模块命名为pd
4  import numpy as np                     # 导入numpy
       模块命名为np
5  import matplotlib.pyplot as plt        # 导入
       matplotlib.pyplot模块命名为plt
6
7  #参数
8  w=2.2143# 入射波浪频率 (s-1)
9  f=4890# 垂荡激励力振幅 (N)
10 #f=5890# 垂荡激励力振幅 (N)
11 T=2*pi/w #周期
12 dt=0.02# 时间步长
13 m3=1165.992# 垂荡附加质量 (kg)
14 k1=167.8395# 垂荡兴波阻尼系数 (N·s/m)
15 m1=4866# 浮子质量 (kg)
16 r1=1# 浮子底半径 (m)
17 h11=3# 浮子圆柱部分高度 (m)
18 h12=0.8# 浮子圆锥部分高度 (m)
19 m2=2433# 振子质量 (kg)
20 r2=0.5# 振子半径 (m)
21 h2=0.5# 振子高度 (m)
22 rho=1025# 海水的密度 (kg/m3)
23 g=9.8# 重力加速度 (m/s2)
```

```
24  F_gangdu1=80000# 弹簧刚度 (N/m)
25  L=0.5# 弹簧原长 (m)
26  F_gangdu2=250000# 扭转弹簧刚度 (N·m)
27
28  S=pi*r1**2# 浮子横截面积
29  V0=(m1+m2)/rho # 静止时的排水体积
30  zeta=10000# 阻尼器阻尼系数
31
32  t=np.arange(0,40*T,dt) #时间
33  n=len(t)
34  v1=np.zeros(n, dtype=float)
35  x1=np.zeros(n, dtype=float)
36  v2=np.zeros(n, dtype=float)
37  x2=np.zeros(n, dtype=float)
38  W=np.zeros(n, dtype=float)
39  v1[0]=0# 浮子初始位移
40  x1[0]=0# 浮子初始速度
41  v2[0]=0# 振子初始位移
42  x2[0]=0# 振子初始速度
43  W[0]=0# 初始功率
44  # 初始化基本参数
45  def initParameter():
46  t_init = 100      # 初始退火温度
47  t_final = 0              # 终止退火温度
48  n_1 = 70                 # 内循环运行次数
49  return t_init,t_final,n_1
```

```
50                  # Metropolis准则判断是否接受新的阻尼系数
51                  def Metropolis(curr_power,prev_power,best_power
                        ,prev_x,curr_x,best_x,t_now):
52                  # dE新解与原解的差值
53                  dE = curr_power - prev_power
54                  # 新的阻尼系数对应的平均功率大于当前解，接受新
                        解
55                  if dE > 0:
56                  accept = True
57                  # 新的阻尼系数对应的平均功率大于最优解，将新解
                        保存为最优解
58                  if curr_power > best_power:
59                  best_x[:] = curr_x[:]
60                  best_power = curr_power
61                  # 新的阻尼系数对应的平均功率小于当前解，以一定
                        概率接受新解
62                  else:
63                  # 依据Metropolis 准则计算接受概率
64                  p_accept = exp(-dE / t_now)
65                  if p_accept > random.random():
66                  accept = True
67                  else:
68                  accept = False
69                  # 接受新解，将新解保存为当前解
70                  if accept == True:
71                  prev_x[:] = curr_x[:]
```

```
72                  prev_power = curr_power
73                  return prev_power,prev_x,best_power,best_x
74                  #生成新的决策变量
75                  def random_new(k):#k为决策变量列表
76                  if k[0]<=2000:
77                  k1 = k[0] + random.uniform(0, 1000)
78                  elif k[0]>=98000:
79                  k1 = k[0] + random.uniform(-1000, 0)
80                  else:
81                  k1=k[0]+random.uniform(-1000, 1000)
82                  if k[1]<=0.02:
83                  k2 = k[1] + random.uniform(0, 0.01)
84                  elif k[0]>=0.98:
85                  k2 = k[1] + random.uniform(-0.01, 0)
86                  else:
87                  k2 = k[1] + random.uniform(-0.01, 0.01)
88
89                  return [k1,k2]
90
91                  # 设置基本参数
92                  t_init,tFinal,n_1 = initParameter()
93                  # 设置初始解（当前、最优）
94                  t_now  = t_init     #初始化 当前温度
95                  #初始化当前决策参数(比例系数[0,100000] 幂指数
                        [0,1] )
96                  prev_x=[20000,0.5]
```

```
97          #初始化最优决策参数
98          best_x=prev_x.copy()
99          #计算当前平均输出功率(目标函数)
100         prev_power=W[0]
101         #初始化最大输出功率(目标函数)
102         best_power=prev_power
103         best_power_record = []        # 初始化最大输出功
                率记录表
104         # 模拟退火
105         # 终止条件
106         while t_now >= tFinal:
107         # 当前温度下, 寻找最优阻尼系数
108         for j in range(n_1):
109         # 随机产生新的阻尼系数
110         curr_x=random_new(prev_x)
111         #计算目标函数average power
112         for i in range(1, n):
113         # zeta1=current_x[0]*abs(v1[i - 1] - v2[i - 1])
                **current_x[1]
114         zeta1 = curr_x[0]
115         F_1 = -rho * g * (S * x1[i - 1])  # 静水恢复力
116         F_2 = -F_gangdu1 * (x1[i - 1] - x2[i - 1])  #
                弹力
117         F_3 = -zeta1 * (v1[i - 1] - v2[i - 1])  # 阻尼
                器阻力
118         F_4 = -k1 * v1[i - 1]  # 兴波阻尼力
```

```
119                 F_5 = f * cos(w * i * dt)  # 波浪激励力
120                 F_h1 = F_1 + F_2 + F_3 + F_4 + F_5  # 浮子受力
121                 F_h2 = -F_2 - F_3  # 振子受力
122                 v1[i] = dt / (m1 + m3) * F_h1 + v1[i - 1]  # 浮
                        子速度
123                 v2[i] = dt / m2 * F_h2 + v2[i - 1]  # 振子速度
124                 x1[i] = 0.5 * (v1[i] + v1[i - 1]) * dt + x1[i -
                        1]  # 浮子位移
125                 x2[i] = 0.5 * (v2[i] + v2[i - 1]) * dt + x2[i -
                        1]  # 振子位移
126                 W[i] = abs(F_3 * ((x1[i] - x1[i - 1]) - (x2[i]
                        - x2[i - 1])))
127                 W_aver = sum(W)/(40*T)
128                 print('W_aver',W_aver,curr_x[0])
129
130                 #判断是否接受新的阻尼系数
131                 curr_power = W_aver
132                 #print('current_power',current_power)
133                 prev_power,prev_x,best_power,best_x=Metropolis(
                        curr_power, prev_power, best_power,prev_x,
                        curr_x, best_x, t_now)
134                 # 结束在当前温度的搜索, 更新最大平均功率列表
135                 best_power_record.append(best_power)
                                # 将本次温度下的最大输出功率加入
                        记录表
136                 #curr_power_record.append(prev_power)
```

```
                t_now = t_now -5
                print(max(best_power_record))
                print(best_x)
                itter=[]
                for i in range(21):
                itter.append(i)
                plt.rcParams['font.sans-serif']=['SimHei']
                plt.rcParams['axes.unicode_minus']=False
                plt.figure()
                plt.plot(itter,best_power_record)

                plt.xlabel('迭代次数')
                plt.ylabel('最优解')
                plt.title('模拟退火求最优解过程')
                plt.show()
```

问题二第二问 python 程序:

```
from math import *                      # 导入math模块
import random                           # 导入random模块
import pandas as pd                     # 导入pandas模块命
    名为pd
import numpy as np                      # 导入numpy模块命名
    为np
import matplotlib.pyplot as plt         # 导入matplotlib.
    pyplot模块命名为plt

#参数
```

```
8   w=2.2143# 入射波浪频率 (s-1)
9   f=4890# 垂荡激励力振幅 (N)
10  T=2*pi/w #周期
11  dt=0.02# 时间步长
12  m3=1165.992# 垂荡附加质量 (kg)
13  k1=167.8395# 垂荡兴波阻尼系数 (N·s/m)
14  m1=4866# 浮子质量 (kg)
15  r1=1# 浮子底半径 (m)
16  h11=3# 浮子圆柱部分高度 (m)
17  h12=0.8# 浮子圆锥部分高度 (m)
18  m2=2433# 振子质量 (kg)
19  r2=0.5# 振子半径 (m)
20  h2=0.5# 振子高度 (m)
21  rho=1025# 海水的密度 (kg/m3)
22  g=9.8# 重力加速度 (m/s2)
23  F_gangdu1=80000# 弹簧刚度 (N/m)
24  L=0.5# 弹簧原长 (m)
25  F_gangdu2=250000# 扭转弹簧刚度 (N·m)
26  S=pi*r1**2# 浮子横截面积
27
28  t=np.arange(0,40*T,dt) #时间
29  n=len(t)
30  v1=np.zeros(n, dtype=float)
31  x1=np.zeros(n, dtype=float)
32  v2=np.zeros(n, dtype=float)
33  x2=np.zeros(n, dtype=float)
```

```
34          P=np.zeros(n, dtype=float)
35          zetazeta=np.zeros(n, dtype=float)
36
37          # 初始化控制参数
38          def init_parameter():
39          t_init = 100                # 初始退火温度
40          t_final = 0                 # 终止退火温度
41          n_1 = 100                   # 内循环运行次数
42          return t_init,t_final,n_1
43
44          # 按照 Metropolis 准则决定是否接受新的阻尼系数
45          def Metropolis(curr_power,prev_power,best_power,prev_x,
                curr_x,best_x,tNow):
46          # dE 新解与原解的差值
47          dE = curr_power - prev_power
48          # 如果新的阻尼系数对应的平均功率好于当前解，则接受新的
                阻尼系数
49          if dE > 0:
50          accept = True
51          # 如果新的阻尼系数的目标函数好于最优解，则将新解保存为
                最优解
52          if curr_power > best_power:
53          best_x[:] = curr_x[:]
54          best_power = curr_power
55          # 如果的阻尼系数的目标函数比当前解差，则以一定概率接受
                新的阻尼系数
```

```
56          else:
57          # 按照Metropolis 判断是否接受新的阻尼系数
58          p_accept = exp(-dE / tNow)
59          if p_accept > random.random():
60          accept = True
61          else:
62          accept = False
63          # 接受新的阻尼系数, 并将新解保存为当前解
64          if accept == True:
65          prev_x[:] = curr_x[:]
66          prev_power = curr_power
67          return prev_power,prev_x,best_power,best_x
68
69          #生成新的决策变量
70          def random_new(k):#k为决策变量列表
71          if k[0]<=4000:
72          k1 = k[0] + random.uniform(0, 2000)
73          elif k[0]>=96000:
74          k1 = k[0] + random.uniform(-2000, 0)
75          else:
76          k1=k[0]+random.uniform(-2000, 2000)
77
78          if k[1]<=0.04:
79          k2 = k[1] + random.uniform(0, 0.02)
80          elif k[1]>=0.96:
81          k2 = k[1] + random.uniform(-0.02, 0)
```

```
82          else:
83          k2=k[1]+random.uniform(-0.02, 0.02)
84          return [k1,k2]
85
86          # 设置控制参数
87          t_init,t_final,n_1= init_parameter()
88          # 初始化参数（当前、最优）
89          v1[0]=0# 浮子初始位移
90          x1[0]=0# 浮子初始速度
91          v2[0]=0# 振子初始位移
92          x2[0]=0# 振子初始速度
93
94          t_now = t_init     #初始化 当前温度
95          #初始化当前决策参数(比例系数[0,100000] 幂指数 [0,1] )
96          prev_x=[50000,0.5]
97          #初始化最优决策参数
98          best_x=prev_x.copy()
99          #初始化当前平均输出功率(目标函数)
100         P[0]=0
101         prev_power=P[0]
102         #初始化最大平均输出功率(目标函数)
103         best_power=prev_power
104         #不同温度下的最大平均功率记录
105         best_power_record = []
106         best_k=[0,0]
107         # 模拟退火
```

```
108         # 终止条件
109         while t_now >= t_final:
110         # 当前温度下, 求出最大输出功率
111         for k in range(n_1):
112         # 随机产生新的阻尼系数
113         curr_x=random_new(prev_x)
114         #计算目标函数average power
115         for i in range(1, n):
116         zeta1=curr_x[0]*abs(v1[i - 1] - v2[i - 1])**curr_x[1]
117         zetazeta[i] = zeta1
118         F_1 = -rho * g * (S * x1[i - 1]) # 静水恢复力
119         F_2 = -F_gangdu1 * (x1[i - 1] - x2[i - 1]) # 弹力
120         F_3 = -zeta1 * (v1[i - 1] - v2[i - 1]) # 阻尼器阻力
121         F_4 = -k1 * v1[i - 1] # 兴波阻尼力
122         F_5 = f * cos(w * i * dt) # 波浪激励力
123         F_h1 = F_1 + F_2 + F_3 + F_4 + F_5 # 浮子受力
124         F_h2 = -F_2 - F_3 # 振子受力
125         v1[i] = dt / (m1 + m3) * F_h1 + v1[i - 1] # 浮子速度
126         v2[i] = dt / m2 * F_h2 + v2[i - 1] # 振子速度
127         x1[i] = 0.5 * (v1[i] + v1[i - 1]) * dt + x1[i - 1] #
              浮子位移
128         x2[i] = 0.5 * (v2[i] + v2[i - 1]) * dt + x2[i - 1] #
              振子位移
129         P[i] = abs(F_3 * ((x1[i] - x1[i - 1]) - (x2[i] - x2[i -
              1])))
130         P_aver=np.mean(P)
```

```
131         print('P_aver',P_aver,curr_x)
132         #判断是否接受新的阻尼系数
133         curr_power = P_aver
134         prev_power,prev_x,best_power,best_x=Metropolis(
                curr_power, prev_power, best_power,prev_x, curr_x,
                best_x, t_now)
135
136         # 完成当前温度的搜索, 更新最优解列表
137         best_power_record.append(best_power)                    #
                将本次温度下的最大平均功率加入记录表
138         t_now = t_now -5
139         print(max(best_power_record))
140         print(best_x)
141
142         itter=[]
143
144         for i in range(1,22):
145         itter.append(i)
146         plt.rcParams['font.sans-serif']=['SimHei']
147         plt.rcParams['axes.unicode_minus']=False
148         plt.figure()
149         plt.plot(itter,best_power_record)
150
151         plt.xlabel('迭代次数')
152         plt.ylabel('最优解')
153         plt.title('模拟退火求最优解过程')
```

```
154         plt.show()
```

问题三 python 程序:

```
1   from math import *                          # 导入math
        模块
2   import random                               # 导入
        random模块
3   import pandas as pd                         # 导入
        pandas模块命名为pd
4   import numpy as np                          # 导入numpy
        模块命名为np
5   import matplotlib.pyplot as plt             # 导入
        matplotlib.pyplot模块命名为plt
6
7   ####参数
8   #振子 浮子 弹簧的参数
9   rho=1025# 海水的密度 (kg/m3)
10  g=9.8# 重力加速度 (m/s2)
11  m1=4866# 浮子质量 (kg)
12  m2=2433# 振子质量 (kg)
13  m3=1028.876# 垂荡附加质量 (kg)
14  r1=1# 浮子底半径 (m)
15  r2=0.5# 振子半径 (m)
16  h11=3# 浮子圆柱部分高度 (m)
17  h12=0.8# 浮子圆锥部分高度 (m)
18  h2=0.5# 振子高度 (m)
19  k=80000# 弹簧刚度 (N/m)
```

```
20  l00=0.5# 弹簧原长 (m)
21  l0=l00-m2*g/k#弹簧初始长度
22  #参数
23  alpha1=10000#直线阻尼器的阻尼系数N·s/m
24  alpha2=1000 #旋转阻尼器的阻尼系数N·m·s
25  k1=683.4558# 垂荡兴波阻尼系数 (N·s/m)
26  k2=654.3383# 纵摇兴波阻尼系数 (N·m·s)
27  k3=8890.7# 静水恢复力矩系数 (N·m)
28  k4=250000# 扭转弹簧刚度 (N·m)
29  I1=31961#浮子的转动惯量(kg·m2)
30  I3=7001.914# 浮子的附加转动惯量(kg·m2)# 纵摇附
        加转动惯量
31
32  #f=3640# 垂荡激励力振幅 (N)
33  f=4640# 垂荡激励力振幅 (N)
34
35  omega=1.7152# 入射波浪频率 (s-1)
36  T=2*pi/omega #周期
37  dt=0.002# 时间步长
38  #L=1690# 纵摇激励力矩振幅 (N·m)
39  L=1990# 纵摇激励力矩振幅 (N·m)
40
41  V0=(m1+m2)/rho# 初始排水体积
42  x_1 = 2.177
43  t=np.arange(0,40*T,dt) #时间
44  n=len(t)
```

```
45                  v1=np.zeros(n, dtype=float)
46                  x1=np.zeros(n, dtype=float)
47                  v2=np.zeros(n, dtype=float)
48                  x2=np.zeros(n, dtype=float)
49                  omega1=np.zeros(n, dtype=float)
50                  theta1=np.zeros(n, dtype=float)
51                  omega2=np.zeros(n, dtype=float)
52                  theta2=np.zeros(n, dtype=float)
53                  l=np.zeros(n, dtype=float)
54                  v1[0]=0# 浮子初始速度
55                  x1[0]=0# 浮子初始位移
56                  v2[0]=0# 振子初始速度
57                  x2[0]=0# 振子初始位移
58                  l[0]=l0# 弹簧初始长度
59                  omega1[0]=0# 浮子初始角速度
60                  theta1[0]=0# 浮子初始角位移
61                  omega2[0]=0# 振子初始角速度
62                  theta2[0]=0# 振子初始角位移
63
64
65                  for i in range(1,n):
66                  z = 2.8 - x1[i - 1]
67                  V=(49/15-3.8+z/cos(theta1[i-1]))*pi
68                  F_1 = rho * g * (V - V0)  # 静水恢复力
69                  F_2 = k * (l[i - 1] - l0)  # 弹力
70                  F_3 = -alpha1 * (v1[i - 1] * cos(theta2[i - 1])
```

```
        - v2[i - 1]) # 阻尼器阻力
71      F_4 = -k1 * v1[i - 1] # 兴波阻尼力
72      F_5 = f * cos(omega * i * dt) # 波浪激励力
73      F_6 = m2 * g * (1 - cos(theta2[i - 1]))#重力变
            化
74      F_h1 = F_1 + (F_2 + F_3)*cos(theta1[i-1]) + F_4
            + F_5 # 浮子受力
75      F_h2 = -F_2 - F_3 + F_6 # 振子受力
76      v1[i] = dt / (m1 + m3) * F_h1 + v1[i - 1] # 浮
            子速度
77      v2[i] = dt / m2 * F_h2 + v2[i - 1] # 振子速度
78      x1[i] = 0.5 * (v1[i] + v1[i - 1]) * dt + x1[i -
            1] # 浮子位移
79      x2[i] = 0.5 * (v2[i] + v2[i - 1]) * dt + x2[i -
            1] # 振子位移
80      x_2 = 0.25 + l[i-1]#x2
81      M1 = -k3 * theta1[i - 1] # 静水恢复力矩
82      M2 = -k4 * (theta1[i - 1] - theta2[i - 1]) #
            扭转弹簧力矩
83      M3 = -alpha2 * (omega1[i - 1] - omega2[i - 1])
            # 选择阻尼器力矩
84      M4 = -k2 * omega1[i-1] # 兴波阻尼力矩
85      M5 = L * cos(omega * i * dt) # 波浪激励力矩
86      M6 = m1 * g * x_1 * sin(theta1[i - 1]) #* sgn(
            theta1[i - 1]) # 浮子重力矩
87      M7 = m2 * g * x_2 * sin(theta2[i - 1]) #* sgn(
```

```
    theta2[i - 1]) # 振子重力矩
I2 = m2 * (0.25 + l[i-1]) ** 2 + m2 / 12 * (3 *
    0.25 + 0.25) # 振子的转动惯量
omega1[i]=dt/(I1+I3)*(M1+M2+M3+M4+M5+M6)#浮子角
    速度
omega2[i]=dt/I2*(-M2-M3+M7)#振子角速度
theta1[i]=(omega1[i]+omega1[i-1])/2*dt + theta1
    [i-1]#浮子角位移
theta2[i] = (omega2[i]+omega2[i-1])/2 * dt +
    theta2[i - 1]#振子角位移
l[i]=l[i-1]+(x2[i]-x2[i-1])-((x1[i]-x1[i-1])
    +0.8*(cos(theta1[i])-cos(theta1[i-1])))/cos\
(theta2[i - 1])#弹簧长度
if i==5000 or i==10000 or i==20000 or i==30000
    or i==50000:
print(x2[i],v2[i],theta2[i],omega2[i])

plt.figure()
plt.plot(t,omega2)
plt.plot(t,theta2)
plt.figure()
plt.plot(t,omega1)
plt.plot(t,theta1)
plt.figure()
plt.plot(t,v1)
plt.plot(t,x1)
```

```
107                 plt.figure()
108                 plt.plot(t,v2)
109                 plt.plot(t,x2)
110
111                 plt.show()
```

问题四 python 程序:

```
1                   from math import *                  # 导入math
                        模块
2                   import random                       # 导入
                        random模块
3                   import pandas as pd                 # 导入
                        pandas模块命名为pd
4                   import numpy as np                  # 导入numpy
                        模块命名为np
5                   import matplotlib.pyplot as plt     # 导入
                        matplotlib.pyplot模块命名为plt
6
7                   ####参数
8                   #振子 浮子 弹簧的参数
9                   m1=4866# 浮子质量 (kg)
10                  m2=2433# 振子质量 (kg)
11                  m3=1091.099# 垂荡附加质量 (kg)
12                  r1=1# 浮子底半径 (m)
13                  r2=0.5# 振子半径 (m)
14                  h11=3# 浮子圆柱部分高度 (m)
15                  h12=0.8# 浮子圆锥部分高度 (m)
```

```
16  h2=0.5# 振子高度 (m)
17  F_gangdu1=80000# 弹簧刚度 (N/m)
18  l_=0.5# 弹簧原长 (m)
19  l0=l_-m2/F_gangdu1#弹簧初始长度
20  rho=1025# 海水的密度 (kg/m3)
21  g=9.8# 重力加速度 (m/s2)
22
23  #系数
24  k1=528.5018# 垂荡兴波阻尼系数 (N·s/m)
25  k2=1655.909# 纵摇兴波阻尼系数 (N·m·s)
26  k3=8890.7# 静水恢复力矩系数 (N·m)
27  k4=250000# 扭转弹簧刚度 (N·m)
28
29  #转动惯量
30  I1=31961#浮子的转动惯量(kg·m2)
31  I3=7142.493# 浮子的附加转动惯量 纵摇附加转动惯
        量 (kg·m2)
32
33  f=1760# 垂荡激励力振幅 (N)
34  omega=1.9806# 入射波浪频率 (s-1)
35  T=2*pi/omega #周期
36  dt=0.002# 时间步长
37  L=2140# 纵摇激励力矩振幅 (N·m)
38  V0=(m1+m2)/rho# 初始排水体积
39  x_1 = 2.177
40  t=np.arange(0,20*T,dt) #时间
```

```
41                  n=len(t)
42                  v1=np.zeros(n, dtype=float)
43                  x1=np.zeros(n, dtype=float)
44                  v2=np.zeros(n, dtype=float)
45                  x2=np.zeros(n, dtype=float)
46                  omega1=np.zeros(n, dtype=float)
47                  theta1=np.zeros(n, dtype=float)
48                  omega2=np.zeros(n, dtype=float)
49                  theta2=np.zeros(n, dtype=float)
50                  l=np.zeros(n, dtype=float)
51                  P=np.zeros(n, dtype=float)
52                  P1=np.zeros(n, dtype=float)
53
54                  # 设置基本参数
55                  def init_parameter():
56                  t_init = 100              # 初始退火温度
57                  t_final = 0               # 终止退火温度
58                  n_1 = 100                 # 内循环运行次数
59                  return t_init,t_final,n_1
60
61                  # Metropolis准则判断是否接受新的阻尼系数
62                  def Metropolis(curr_power,prev_power,best_power
                        ,prev_x,curr_x,best_x,t_now):
63                  # dE 新解与原解的差值
64                  dE = curr_power - prev_power
65                  # 新的阻尼系数对应的平均功率大于当前解, 接受新
```

```
        解
66      if dE > 0:
67      accept = True
68      # 新的阻尼系数对应的平均功率值大于最大平均功
            率，将新的阻尼系数保存为最优解
69      if curr_power > best_power:
70      best_x[:] = curr_x[:]
71      best_power = curr_power
72      # 新的阻尼系数对应的平均功率小于当前平均功率，
            以一定概率接受新的阻尼系数
73      else:
74      # 依据Metropolis准则计算接受的概率
75      p_accept = exp(-dE / t_now)
76      if p_accept > random.random():
77      accept = True
78      else:
79      accept = False
80      # 接受新的阻尼系数，将新解保存为当前解
81      if accept == True:
82      prev_x[:] = curr_x[:]
83      prev_power = curr_power
84      return prev_power,prev_x,best_power,best_x
85
86      #生成新的决策变量
87      def random_new(k):#k为决策变量列表
88      if k[0]<=4000:
```

```
89                  k1 = k[0] + random.uniform(0, 2000)
90                  elif k[0]>=96000:
91                  k1 = k[0] + random.uniform(-2000, 0)
92                  else:
93                  k1=k[0]+random.uniform(-2000, 2000)
94
95                  if k[1]<=4000:
96                  k2 = k[1] + random.uniform(0, 2000)
97                  elif k[1]>=96000:
98                  k2 = k[1] + random.uniform(-2000, 0)
99                  else:
100                 k2=k[1]+random.uniform(-2000, 2000)
101                 return [k1,k2]
102
103                 # 设置基本参数
104                 t_init,t_final,n_1= init_parameter()
105                 # 初始化参数（当前、最优）
106                 v1[0]=0# 浮子初始位移
107                 x1[0]=0# 浮子初始速度
108                 v2[0]=0# 振子初始位移
109                 x2[0]=0# 振子初始速度
110                 omega1[0]=0# 浮子初始角速度
111                 theta1[0]=0# 浮子初始角位移
112                 omega2[0]=0# 振子初始角速度
113                 theta2[0]=0# 振子初始角位移
114                 t_now  = t_init     #初始化 当前温度
```

```
115                 #初始化当前决策参数(比例系数[0,100000] 幂指数
                        [0,1] )
116                 prev_x=[50000,50000]
117                 #初始化最优决策参数
118                 best_x=prev_x.copy()
119                 #初始化当前平均输出功率(目标函数)
120                 P[0]=0
121                 P1[0]=0
122                 prev_power=P[0]
123                 #初始化最大平均输出功率(目标函数)
124                 best_power=prev_power
125                 #不同温度下的最大平均功率记录
126                 best_power_record = [0]
127                 # 模拟退火
128                 aa=0
129                 # 终止条件
130                 while t_now >= t_final:
131                 # 当前温度下, 寻找最优阻尼系数
132                 for k in range(n_1):
133                 # 随机产生新的阻尼系数
134                 curr_x=random_new(prev_x)
135                 alpha1 = curr_x[0]  # 直线阻尼器的阻尼系数N·s/m
136                 alpha2 = curr_x[1]  # 旋转阻尼器的阻尼系数N·m·s
137                 #计算目标函数average power
138                 for i in range(1, n):
139                 z = 2.8 - x1[i - 1]
```

```
140                 V = (49 / 15 - 3.8 + z / cos(theta1[i - 1])) *
                        pi
141                 F_1 = rho * g * (V - V0)  # 静水恢复力
142                 F_2 = k * (l[i - 1] - l0)  # 弹力
143                 F_3 = -alpha1 * (v1[i - 1] * cos(theta2[i - 1])
                        - v2[i - 1])  # 阻尼器阻力
144                 F_4 = -k1 * v1[i - 1]  # 兴波阻尼力
145                 F_5 = f * cos(omega * i * dt)  # 波浪激励力
146                 F_6 = m2 * g * (1 - cos(theta2[i - 1]))  # 重力
                        变化
147                 F_h1 = F_1 + (F_2 + F_3) * cos(theta1[i - 1]) +
                        F_4 + F_5  # 浮子受力
148                 F_h2 = -F_2 - F_3 + F_6  # 振子受力
149                 v1[i] = dt / (m1 + m3) * F_h1 + v1[i - 1]  # 浮
                        子速度
150                 v2[i] = dt / m2 * F_h2 + v2[i - 1]  # 振子速度
151                 x1[i] = 0.5 * (v1[i] + v1[i - 1]) * dt + x1[i -
                        1]  # 浮子位移
152                 x2[i] = 0.5 * (v2[i] + v2[i - 1]) * dt + x2[i -
                        1]  # 振子位移
153
154                 x_2 = 0.25 + l[i - 1]  # x2
155                 M1 = -k3 * theta1[i - 1]  # 静水恢复力矩
156                 M2 = -k4 * (theta1[i - 1] - theta2[i - 1])  #
                        扭转弹簧力矩
157                 M3 = -alpha2 * (omega1[i - 1] - omega2[i - 1])
```

```
    # 选择阻尼器力矩
158 M4 = -k2 * omega1[i - 1]  # 兴波阻尼力矩
159 M5 = L * cos(omega * i * dt)  # 波浪激励力矩
160 M6 = m1 * g * x_1 * sin(theta1[i - 1])  # * sgn
    (theta1[i - 1])  # 浮子重力矩
161 M7 = m2 * g * x_2 * sin(theta2[i - 1])  # * sgn
    (theta2[i - 1])  # 振子重力矩
162 I2 = m2 * (0.25 + l[i - 1]) ** 2 + m2 / 12 * (3
     * 0.25 + 0.25)  # 振子的转动惯量
163 omega1[i] = dt / (I1 + I3) * (M1 + M2 + M3 + M4
     + M5 + M6)  # 浮子角速度
164 omega2[i] = dt / I2 * (-M2 - M3 + M7)  # 振子角
    速度
165 theta1[i] = omega1[i] * dt + theta1[i - 1]  #
    浮子角位移
166 theta2[i] = omega2[i] * dt + theta2[i - 1]  #
    振子角位移
167 l[i] = l[i - 1] + (x2[i] - x2[i - 1]) - (
168 (x1[i] - x1[i - 1]) + 0.8 * (cos(theta1[i]) -
    cos(theta1[i - 1]))) / cos \
169 (theta2[i - 1])  # 弹簧长度
170 P[i] = abs(F_3 * ((x1[i] - x1[i - 1]) - (x2[i]
    - x2[i - 1])))
171 P1[i] = abs(M3 * ((theta1[i] - theta1[i - 1]) -
     (theta2[i] - theta2[i - 1])))
172
```

```
173                 P_aver=sum(P)/(20 * T)
174                 P1_aver = sum(P1)/(20 * T)
175                 P_P1_aver = sum(P + P1)/(20 * T)
176                 print('P_aver',P_aver,P1_aver,P_P1_aver,curr_x)
177
178                 #判断是否接受新的阻尼系数
179                 curr_power = P_aver
180                 prev_power,prev_x,best_power,best_x=Metropolis(
                        curr_power, prev_power, best_power,prev_x,
                        curr_x, best_x, t_now)
181                 # 结束在当前温度的搜索, 更新最大平均功率列表
182                 best_power_record.append(best_power)          #
                        本次温度下的最大输出功率加到最大功率记录表
183                 t_now = t_now -1
184                 print(max(best_power_record))
185                 print(best_x)
```