波浪能最大输出功率优化设计

浙江大学　赵云鹏 何逸阳 闻人豪昱

摘要

波浪能作为一种海洋可再生能源，在可再生能源产业中起着重要的作用。本文研究了一种波浪能转换装置, 建立了由浮子和振子组成的动力学系统模型, 并针对不同情况进行**解析解**和数值解的计算。此外, 根据模型求解结果以最大化平均功率为目标获得相关参数的最优值, 并对其做敏感度分析以获得参数的较优区间.

针对问题一、二，我们首先建立了浮子和振子一维垂荡运动的运动模型, 并对二者分别进行受力分析得到运动方程。在阻尼系数恒定的情况下, 方程为二阶常系数线性微分方程组, 我们通过**拉普拉斯变换**的方式求得方程在像空间的解, 再使用 **Mathematica** 通过有理分式反演法进行逆变换得到方程的解。阻尼系数不恒定时，微分方程组包含非线性项，较难求出解析解。为了求解这个初值问题，我们在分析了方程的刚性后, 选择使用隐式 **Runge-Kutta** 方法求得数值解。求得数值解后通过**辛普森法**积分计算功率，进而通过**下山单纯形法**以最大输出功率为目标对阻尼器的阻尼系数与幂指数进行优化。

针对问题三、四，在同时考虑垂荡的纵摇的情况下, 运动的自由度增加至四个。我们对振子、浮子受力分析，进而得到三个受力方程、两个力矩方程，对应四个坐标以及一个内部力。由于该方程有许多非线性项, 我们选择使用数值方法求解。同样在分析了方程的刚性后, 我们选择使用五阶隐式 **Runge-Kutta** 方法固定步长求解, 进而用**辛普森法**积分，最后用**下山单纯形法**优化。

随后我们通过查阅资料，对所用数值求解方法进行了粗略的精度分析，确定我们的数值算法精度为**四阶精度**，并估计了所用步长下的求解误差，画出了此误差下各个参数最优的粗略范围。

最后，我们就不同大小的激励力振幅做了**敏感度分析**，确定了适用此模型的海浪大小范围。

关键词: 波浪能利用；受迫耦合振动；拉普拉斯变换；Runge-Kutta 法；下山单纯形法

一、问题重述

1.1 问题背景

波浪能作为一种海洋可再生能源，在可再生能源产业中起着重要的作用。本题中，一种波浪能装置利用波浪引起浮子与振子的相对运动，进而驱动阻尼器做功产生能量输出。具体需要解决如下四个问题。

1.2 具体问题重述

第一问：问题一要求建立模型并求解问题。在给定波浪能装置运动结构，仅考虑浮子在竖直方向的垂荡运动的条件下，建立浮子与振子在波浪激励力作用下的运动模型。分别在阻尼系数恒定和阻尼系数与速度成幂次关系两种阻尼模式下，对初始位于平衡状态的浮子和振子应用运动模型进行求解，给出相应时刻的浮子与振子的速度与位移。

第二问：问题二是问题一的优化问题，浮子在波浪中仅作垂荡运动的条件下，分别对问题一的两种阻尼模式建立确定阻尼器最优阻尼系数的数学模型，以 PTO 平均输出功率最大为目标，对比例系数与幂指数进行优化，并计算最大输出功率及相应最优阻尼系数。

第三问：问题三在问题一的基础上加装旋转阻尼器和转轴架，考虑浮子的垂荡和纵摇运动，要求建立浮子与振子同时在波浪激励力和波浪激励力矩作用下的运动模型。对于给定的阻尼系数，求解初始位于平衡状态下，不同时刻的浮子与振子的垂荡位移与速度和纵摇角位移与角速度。

第四问：问题四是问题三的优化问题，在问题三的基础上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，以 PTO 平均输出功率最大为目标，对阻尼系数优化，并计算最大输出功率及相应最优阻尼系数。

二、模型假设

假设 2.1 水面近似保持静止（由于是微幅波，因此水面的上下波动不纳入考虑）；

假设 2.2 假设阻尼对外输出功率转化效率是常数（因此不妨设其为 *1*，不纳入考虑）；

假设 2.3 假设振子质量均匀分布在圆柱体上，并忽略杆的粗细，因此可以把振子近似看成实心圆柱体；

假设 2.4 假设振子完全不与外界接触，只受重力和浮子的影响。同时忽略海风等对浮子的影响；

假设 2.5 不考虑浮子横向上受力（因为其对发电无影响，可通过锚定等方法消除）；

假设 2.6 假设振子、杆以及浮子在受力时不产生形变。

三、符号说明

表 1: 符号说明

符号	说明
m_v	振子质量
m_f	浮子质量
k	弹簧刚度
k_m	扭转弹簧刚度
ν	阻尼器的阻尼系数
m_a	垂荡附加质量
I_a	纵摇附加转动惯量
μ	垂荡兴波阻尼系数
μ_m	纵摇兴波阻尼系数
F	垂荡波浪激励力
L	纵摇激励力矩
r	静水恢复力系数
r_m	静水恢复力矩系数
t_f	迭代终止时间（末时刻）

四、问题分析

4.1 问题一分析

由于题目要求只考虑振子和浮子在竖直方向上的运动，所以这是一个一维二体动力学问题的求解。我们对浮子和振子分别进行受力分析，再通过牛顿第二定律建立运动方程。因为初始状态固定（平衡于水中），外力形式和相互作用力形式已知，我们通过牛顿运动方程来确定任意时刻两物体的运动状态。

1. 阻尼系数为常数：方程为二阶常系数线性微分方程组，可以通过拉普拉斯变换的方式求得方程的解析解，进而确定任意时刻浮子和振子的运动状态。

2. 阻尼系数与浮子和振子的相对速度的绝对值的幂成正比：方程为非线性方程，不易求得解析解。分析得出方程的刚性较大，因此我们通过 5 阶隐式 Runge-Kutta 法固定步长对模型进行求解。

4.2 问题二分析

浮子和振子的垂荡在固定频率的受迫力作用下会趋于稳定，因此我们计算在稳定状态下功率输出的最优解。

1. 在第一小问中, 我们使用复振幅法解析地求出了浮子和振子的稳定运动情况，并计算得到了平均输出功率与阻尼系数 ν 的关系。通过 Mathematica 进行求导得到阻尼系数的最优值。

2. 在第二小问中, 我们选择使用复合 Simpson 方法进行数值积分得到平均功率。在对各个参数下的平均功率作图以及解析分析后，我们发现其局部最优解即全局最优解，进而采用下山单纯形法进行优化求解。

4.3 问题三分析

由于水平方向的运动对输出功率没有影响，可以不纳入考虑, 因此浮子和振子的运动系统共有四个自由度。我们对振子、浮子受力分析，进而获得三个受力方程、两个力矩方程，对应四个坐标以及一个内部力。由于方程具有许多非线性项，求解变得十分困难，几乎无法用解析方式得到结果。因此我们直接采用数值方法。同样由于方程刚性较大，我们通过 5 阶隐式 Runge-Kutta 法固定步长求解。

4.4 问题四分析

同问题二第二小问，我们通过数值积分的方法求得平均功率, 并以最大平均功率为目标对两个阻尼系数进行优化。同样，我们发现局部最优解即为全局最优解，进而采用下山单纯形法进行优化求解。

五、模型建立

5.1 仅考虑垂荡运动的模型

5.1.1 变量、符号说明

表 2: 变量说明

变量	说明
x_v	振子的竖直偏移平衡位置的位移，正方向向上
x_f	浮子的竖直偏移平衡位置的位移，正方向向上
P	瞬时输出功率
Pa	平均输出功率

5.1.2 受力分析

对振子和浮子分别做受力分析，结果如下图所示。

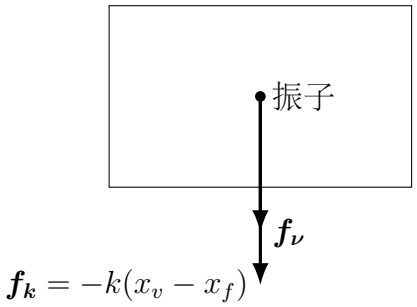

图 1: 振子垂荡受力分析

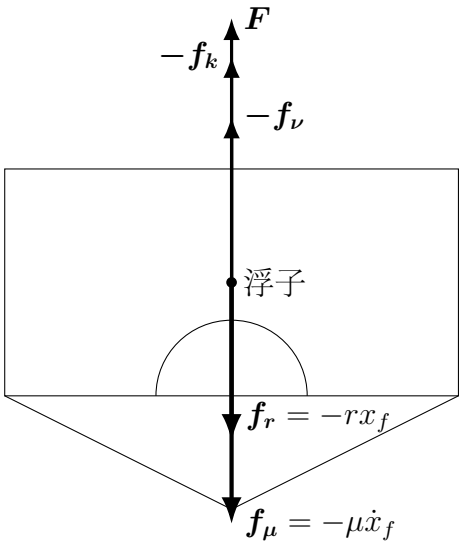

图 2: 浮子垂荡受力分析

振子受力分析

对振子而言，平衡位置处弹簧的弹力与浮子的重力保持平衡，所以在偏移平衡位置后所受的合力为弹簧弹力的改变量 $f_k = -k(x_v - x_f)$ 以及阻尼器的阻力 f_ν, 由牛顿第二定律得出振子运动方程:

$$m_v\ddot{x}_v = -k(x_v - x_f) - \boldsymbol{f_\nu} \tag{1}$$

浮子受力分析

对浮子而言，同样平衡位置的浮力与重力保持平衡，偏移平衡位置后所受的合力为

$$\begin{cases} \text{垂荡波浪激励力} \quad F \\ \text{弹簧的弹力变化量} \quad -f_k = k(x_v - x_f) \\ \text{阻尼器的阻力} \quad -f_\nu \\ \text{垂荡兴波阻尼力} \quad f_\mu = -\mu\dot{x}_f \\ \text{静水恢复力} \quad f_r \end{cases} \tag{2}$$

经计算，浮子漏出水面高度 1m，水下圆柱部分 2m，由后续求解可知浮子振幅小于 1m, 静水恢复力在线性区间（圆柱区间)。因此静水恢复力有以下形式:

$$f_r = -rx_f$$

其中

$$r = \rho g \pi R^2$$

此外，由于推动浮体做摇荡运动的力不仅要推动浮体运动，还要推动浮体周围的流体运动，因而要使浮体在海水中获得加速度，需要施加额外的力，对应牛顿方程中惯性质量的增加 (m_a)。可得:

$$(m_f + m_a)\ddot{x}_f = k(x_v - x_f) + \boldsymbol{f_\nu} - \mu\dot{x}_f + \boldsymbol{F} - rx_f$$

综上，由题目要求得到以下 IVP（初值问题)：

$$\begin{cases} m_v\ddot{x}_v = -k(x_v - x_f) - \boldsymbol{f_\nu} \\ (m_f + m_a)\ddot{x}_f = k(x_v - x_f) + \boldsymbol{f_\nu} - \mu\dot{x}_f + \boldsymbol{F} - rx_f \\ x_v(0) = x_f(0) = \dot{x}_v(0) = \dot{x}_f(0) = 0 \end{cases} \tag{3}$$

5.1.3 功率计算

对于功率的计算，根据题目描述，我们知道功率输出全部来自阻尼器。由假设 2.2，我们得到功率计算公式:

$$P = f_\nu(\dot{x}_v - \dot{x}_f) \tag{4}$$

由于系统稳定时处于振荡状态，因此我们应当求其平均输出功率:

$$Pa = \frac{1}{t_f}\int_0^{t_f} P(t)dt \tag{5}$$

5.2 考虑垂荡和纵摇运动的模型

5.2.1 变量、符号说明

表 3: 变量、符号说明

符号	说明
x_v	振子与浮子的相对位移，正方向向外 (与前文不同)
x_f	浮子的竖直偏移平衡位置的位移，正方向向上
θ_v	振子与浮子圆柱轴线的相对夹角, 顺时针为正
θ_f	浮子圆柱轴线与竖直方向夹角, 顺时针为正
$\overrightarrow{R}$	振子的位矢
$\overrightarrow{i}$	竖直向上基矢
$\overrightarrow{e_r}$	中轴轴向基矢
$\overrightarrow{e_\theta}$	中轴法向基矢
I_v	振子对转轴的转动惯量 (与 x_v 有关)
I_f	浮子对转轴的转动惯量 (常数)

5.2.2 加速度分析

对振子建立牛顿第二定律方程, 需要计算振子的加速度。设振子的位矢为 $\overrightarrow{R}$，有:

$$\overrightarrow{R} = \overrightarrow{x_f} + \overrightarrow{x_v} = x_f \overrightarrow{i} + x_v \overrightarrow{e_r}$$

由几何关系, 我们有:

$$\dot{\overrightarrow{e_r}} = (\dot{\theta}_f + \dot{\theta}_v)\overrightarrow{e_\theta}$$

$$\dot{\overrightarrow{e_\theta}} = -(\dot{\theta}_f + \dot{\theta}_v)\overrightarrow{e_r}$$

$$\overrightarrow{i} = \cos(\theta_f + \theta_v)\overrightarrow{e_r} - \sin(\theta_f + \theta_v)\overrightarrow{e_\theta}$$

因此振子的速度:

$$\begin{aligned}\frac{d\overrightarrow{R}}{dt} &= \dot{x}_f\overrightarrow{i} + \dot{x}_v\overrightarrow{e_r} + x_v\dot{\overrightarrow{e_r}} \\ &= \dot{x}_f\overrightarrow{i} + \dot{x}_v\overrightarrow{e_r} + x_v(\dot{\theta}_f+\dot{\theta}_v)\overrightarrow{e_\theta}\end{aligned}$$

振子的加速度:

$$\begin{aligned}\frac{d^2\overrightarrow{R}}{dt^2} &= \ddot{x}_f\overrightarrow{i} + \ddot{x}_v\overrightarrow{e_r} + \dot{x_v}(\dot{\theta}_f+\dot{\theta}_v)\overrightarrow{e_\theta} + \dot{x_v}(\dot{\theta}_f+\dot{\theta}_v)\overrightarrow{e_\theta} + x_v(\ddot{\theta}_f+\ddot{\theta}_v)\overrightarrow{e_\theta} - x_v(\dot{\theta}_f+\dot{\theta}_v)^2\overrightarrow{e_r} \\ &= \left(\ddot{x}_f\cos(\theta_f+\theta_v) + \ddot{x}_v - x_v(\dot{\theta}_f+\dot{\theta}_v)^2\right)\overrightarrow{e_r} \\ &\quad + \left(-\ddot{x}_f\sin(\theta_f+\theta_v) + 2\dot{x_v}(\dot{\theta}_f+\dot{\theta}_v) + x_v(\ddot{\theta}_f+\ddot{\theta}_v)\right)\overrightarrow{e_\theta}\end{aligned}$$

5.2.3 受力分析

在同时考虑垂荡和纵摇运动情况下, 振子和浮子受力分析如图 3所示.

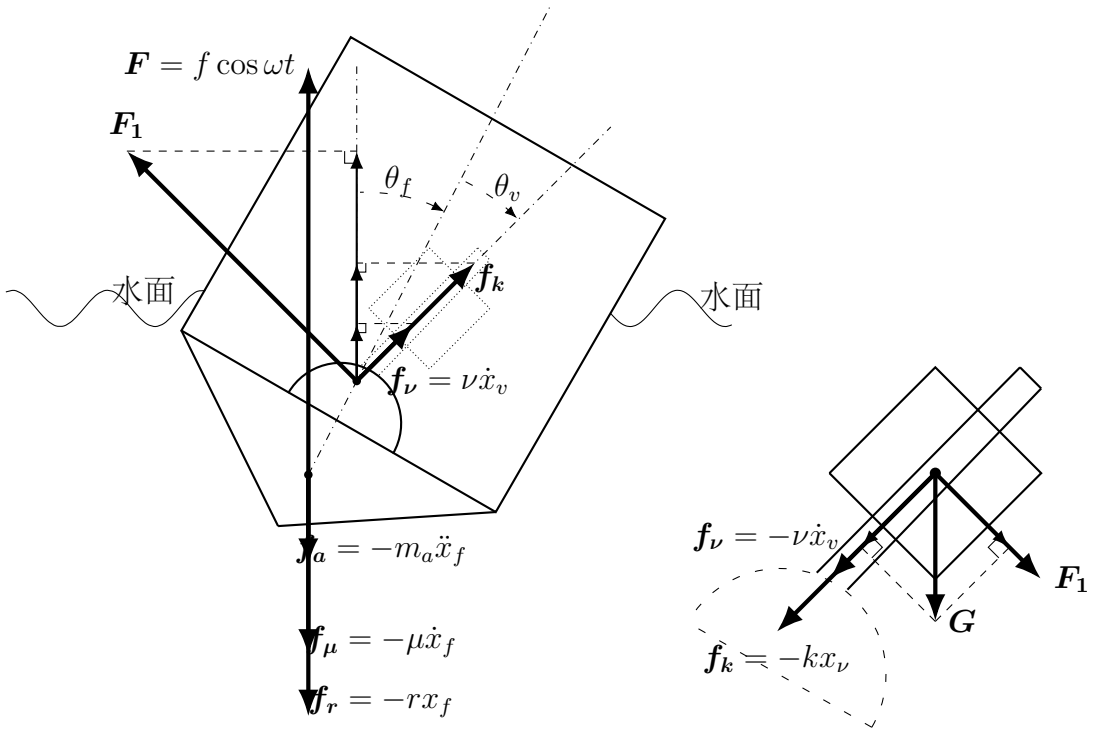

图 3: 振子、浮子受力分析图

振子受力分析

与前文不同, 由于 x_v 并非从平衡位置开始, 因此振子的重力项需单独考虑, 我们设

$$\begin{cases}\text{振子重力} \quad G \\ \text{弹簧弹力} \quad f_k = -kx_\nu \\ \text{阻尼力} \quad f_\nu = -\nu\dot{x}_v \\ \text{振子与浮子垂直于中轴方向的相互作用力} \quad F_1\end{cases} \tag{6}$$

各力方向如图 3所示。

由前文计算 $\overrightarrow{e_r}$ 方向加速度可得

沿中轴方向运动方程:

$$m_v\left(\ddot{x}_f\cos(\theta_f+\theta_v)+\ddot{x}_v-x_v(\dot{\theta}_f+\dot{\theta}_v)^2\right)=-k(x_v-l)-\nu\dot{x}_v-m_vg\cos(\theta_v+\theta_f)$$

垂直中轴方向运动方程:

$$m_v\left(-\ddot{x}_f\sin(\theta_f+\theta_v)+2\dot{x_v}(\dot{\theta}_f+\dot{\theta}_v)+x_v(\ddot{\theta}_f+\ddot{\theta}_v)\right)=F_1+m_vg\sin(\theta_f+\theta_v)$$

浮子受力分析

对于浮子, 只考虑竖直方向的垂荡运动。我们将振子重力项并入弹簧恢复力，即以 $l-m_vg/k$ 为弹簧平衡位置计算弹簧弹力分力 $f_k=k(x_v-l+m_vg/k)\cos(\theta_v+\theta_f)$, 浮子重力项并入静水恢复力 $f_r=-rx_f$ 中, 综上得竖直方向力为

$$\begin{cases}\text{弹簧弹力分力}\quad f_k=k(x_v-l+m_vg/k)\cos(\theta_v+\theta_f)\\ \text{静水恢复力}\quad f_r=-rx_f\\ \text{振子作用力分力}\quad F_1\sin(\theta_v+\theta_f)\\ \text{阻尼力分力}\quad f_\nu=\nu x_v\cos(\theta_v+\theta_f)\\ \text{兴波阻尼力}\quad f_\mu=-\mu\dot{x}_f\\ \text{附加惯性力}\quad fa=-m_a\ddot{x}_f\\ \text{波浪激励力}\quad F=f\cos(\omega t)\end{cases}\tag{7}$$

浮子竖直方向运动方程:

$$m_f\ddot{x}_f=k(x_v-l+m_vg/k)\cos(\theta_v+\theta_f)-rx_f+F_1\sin(\theta_v+\theta_f)+\nu\dot{x}_v\cos(\theta_v+\theta_f)-\mu\dot{x}_f-m_a\ddot{x}_f+f\cos(\omega t)$$

5.2.4 力矩分析

转动惯量计算

首先计算振子对转轴的转动惯量, 均匀圆柱体对经过质心直径的转动惯量为:

$$\frac{m_vr_v^2}{4}+\frac{m_vh_v^2}{12}$$

由平行轴定理可得振子的转动惯量为 $(m_v=2433, r_v=0.5, h_v=0.5)$:

$$I_v=\frac{m_vr^2}{4}+\frac{m_vh^2}{12}+m_vx_v^2$$

随后我们计算浮子外壳对转轴的转动惯量。圆柱、圆锥的高度 $H_1=3, H_2=0.8$, 半径 $R=1$, 圆柱、圆锥的质量分别为 m_1,$m_2(m1+m2=m_f=4866)$, 令 $L=\sqrt{R^2+H_2^2}$ 为圆锥母线长度, 则有:

$$m_1=\frac{2\pi RH_1}{\pi RL+2\pi RH_1}m_f$$

$$m_2 = \frac{\pi RL}{\pi RL + 2\pi RH_1} m_f$$

对圆锥和圆柱壳简单积分可得:

$$I_f = \frac{m_1 R^2}{2} + \frac{m_1 H_1^2}{3} + \frac{m_2 H_2^2}{6} + \frac{m_2 R^2}{4}$$

振子力矩分析

如图 4所示

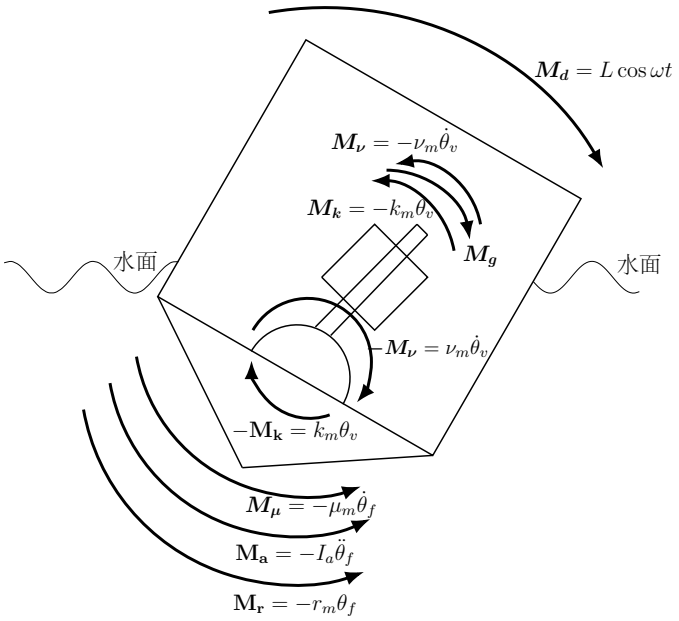

图 4: 振子、浮子力矩分析

振子的力矩来源有三个, 分别为

$$\begin{cases} \text{扭转弹簧的扭矩} \quad M_k = -k_m\theta_v \\ \text{旋转阻尼器的扭矩} \quad M_\nu = -\nu_m\dot{\theta}_v \\ \text{振子重力力矩} \quad M_g = m_v g x_v \sin(\theta_v + \theta_f) \end{cases} \tag{8}$$

由此可得转动方程:

$$-\nu_m\dot{\theta}_v - k_m\theta_v + m_v g x_v \sin(\theta_v + \theta_f) = I_v(\ddot{\theta}_v + \ddot{\theta}_f)$$

浮子力矩分析

浮子的力矩来源分别为

$$\begin{cases} \text{扭转弹簧的扭矩} \quad -M_k = k_m\theta_v \\ \text{旋转阻尼器的扭矩} \quad -M_\nu = \nu_m\dot{\theta}_v \\ \text{静水恢复力矩 (包含重力力矩)} \quad Mr = -r_m\theta_f \\ \text{兴波阻尼力矩} \quad M_\mu = -\mu_m\dot{\theta}_f \\ \text{附加惯性力矩} \quad M_a = -Ia\ddot{\theta}_f \\ \text{波浪激励力矩} \quad M_d = L\cos(\omega t) \end{cases} \tag{9}$$

由此可得转动方程:

$$k_m\theta_v + \nu_m\dot{\theta}_v - r_m\theta_f - \mu_m\dot{\theta}_f - I_a\ddot{\theta}_f + L\cos(\omega t) = I_f\ddot{\theta}_f$$

5.2.5 模型状态方程

综上, 由题给初始条件可得 x_v、x_f、θ_v、θ_f、F_1 所满足的 IVP 为

$$\begin{cases} m_v\left(\ddot{x}_f\cos(\theta_f+\theta_v) + \ddot{x}_v - x_v(\dot{\theta}_f+\dot{\theta}_v)^2\right) = -k(x_v-l) - \nu\dot{x}_v - m_v g\cos(\theta_v+\theta_f) \\ m_v\left(-\ddot{x}_f\sin(\theta_f+\theta_v) + 2\dot{x_v}(\dot{\theta}_f+\dot{\theta}_v) + x_v(\ddot{\theta}_f+\ddot{\theta}_v)\right) = F_1 + m_v g\sin(\theta_f+\theta_v) \\ m_f\ddot{x}_f = k(x_v - l + m_v g/k)\cos(\theta_v+\theta_f) - rx_f + F_1\sin(\theta_v+\theta_f) \\ \qquad\qquad + \nu\dot{x}_v\cos(\theta_v+\theta_f) - \mu\dot{x}_f - m_a\ddot{x}_f + f\cos(\omega t) \\ -\nu_m\dot{\theta}_v - k_m\theta_v + m_v g x_v\sin(\theta_v+\theta_f) = I_v(\ddot{\theta}_v+\ddot{\theta}_f) \\ k_m\theta_v + \nu_m\dot{\theta}_v - r_m\theta_f - \mu_m\dot{\theta}_f - I_a\ddot{\theta}_f + L\cos(\omega t) = I_f\ddot{\theta}_f \\ x_v(0) = l - m_v g/k, \\ x_f(0) = \dot{x}_v(0) = \dot{x}_f(0) = 0 \\ \theta_v(0) = \theta_f(0) = \dot{\theta}_v(0) = \dot{\theta}_f(0) = 0 \end{cases} \tag{10}$$

六、模型求解

6.1 问题一的求解

6.1.1 第一小问

在该小问中，$\boldsymbol{f_\nu}$ 为线性项 $\nu(\dot{x}_v - \dot{x}_f)$，激励力 $\boldsymbol{F} = f\cos(\omega t)$，代入式 (3) 得到二阶线性常系数微分方程组:

$$\begin{cases} m_v\ddot{x}_v = -k(x_v - x_f) - \nu(\dot{x}_v - \dot{x}_f) \\ (m_f + m_a)\ddot{x}_f = k(x_v - x_f) + \nu(\dot{x}_v - \dot{x}_f) - \mu\dot{x}_f - rx_f + f\cos(\omega t) \end{cases} \tag{11}$$

对方程进行拉普拉斯变换得到

$$\begin{cases} m_v p^2 \bar{x}_v(p) = -k(\bar{x}_v(p) - \bar{x}_f(p)) - \nu p(\bar{x}_v(p) - \bar{x}_f(p)) \\ (m_f + m_a)p^2 \bar{x}_f(p) = k(\bar{x}_v(p) - \bar{x}_f(p)) + \nu p(\bar{x}_v(p) - \bar{x}_f(p)) - \mu p \bar{x}_f - r\bar{x}_f(p) + f\frac{p}{p^2+\omega^2} \end{cases} \tag{12}$$

解得

$$\begin{cases} \bar{x}_f(p) = \dfrac{fp\left(k+m_v p^2+pv\right)}{(p^2+w^2)\left((k+m_v p^2+pv)\left(k+p^2(m_a+m_f)+pu+pv+r\right)-(k+pv)^2\right)} \\ \bar{x}_v(p) = \dfrac{fp(k+pv)}{(p^2+w^2)\left(km_a p^2+km_f p^2+km_v p^2+kpu+kr+m_a m_v p^4+m_a p^3 v+m_f m_v p^4+m_f p^3 v+m_v p^3 u+m_v p^3 v+m_v p^2 r+p^2 uv+prv\right)} \end{cases} \tag{13}$$

虽然这个方程较为繁琐, 但是通过有理分式反演法, 利用拉普拉斯变换的基本公式一定能对像函数进行拉普拉斯逆变换。此处我们使用数学软件 Mathematica 进行逆变换, 可求出 $x_f(t), x_v(t)$, 相关代码见附录 B.1.1, 计算结果见 result1-1.xlsx。图 5是求解结果:

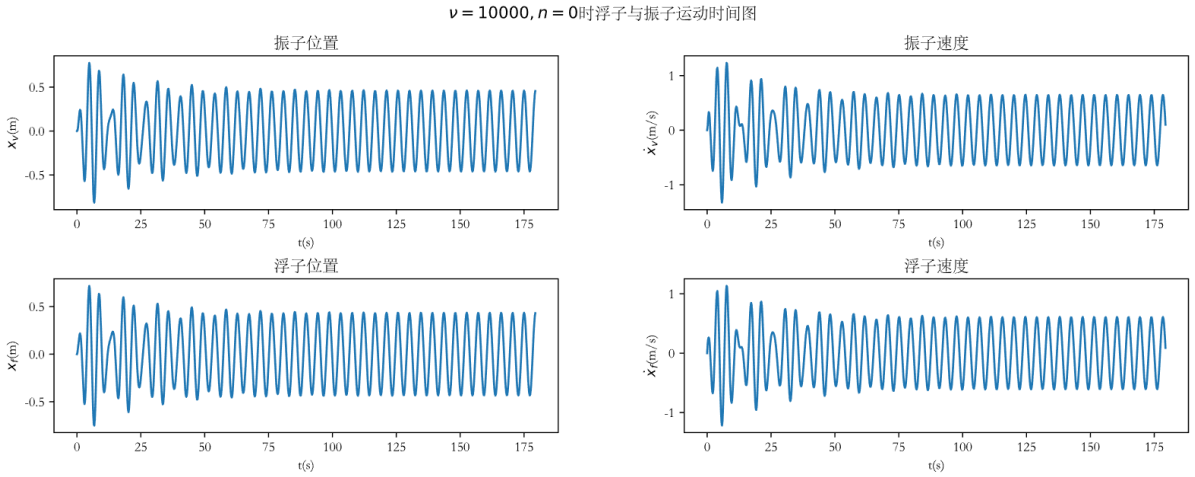

图 5: 振子与浮子运动图

表 5是 10s、20s、40s、60s 和 100s 时刻的垂荡位移和速度:

垂荡位移和速度					
	10s	20s	40s	60s	100s
浮子位移	-0.1907	-0.5907	0.2854	-0.3145	-0.08361
浮子速度	-0.6410	-0.2410	0.3130	-0.4795	-0.6042
振子位移	-0.2117	-0.6342	0.2965	-0.3314	-0.08407
振子速度	-0.6940	-0.2728	0.3329	-0.5157	-0.6430

表 4: 垂荡位移速度表

6.1.2 第二小问

该小问中，$\boldsymbol{f_\nu}$ 为非线性项 $\nu|\dot{x}_v - \dot{x}_f|^n \cdot (\dot{x}_v - \dot{x}_f)$，激励力 $\boldsymbol{F} = f\cos(\omega t)$，代入式 (3) 得到二阶线性常系数微分方程组：

$$\begin{cases} m_v\ddot{x}_v = -k(x_v - x_f) - \nu|\dot{x}_v - \dot{x}_f|^n(\dot{x}_v - \dot{x}_f) \\ (m_f + m_a)\ddot{x}_f = k(x_v - x_f) + \nu|\dot{x}_v - \dot{x}_f|^n(\dot{x}_v - \dot{x}_f) - \mu\dot{x}_f + f\cos(\omega t) - rx_f \end{cases} \tag{14}$$

由于带有非线性项，此方程很难通过解析方式求出精确解，因此我们不得不考虑使用数值方法求解常微分方程。

不难发现，这是一个典型的 ivp 初值问题，因此我们采用初值问题常用的 Runge-Kutta 法（以下称为 RK）。

为了确定方程的刚性程度，我们用 Mathematica 求出了方程的雅可比矩阵 (代码见 B.1.2)，发现矩阵元素之间差异较大（最大比值大于 32.9），因此我们决定采用隐式的 RK 方法 (以下称为 IRK)。

我们注意到 python 中有现成的 IRK 方法（Radau AII），因此我们利用 scipy 库当中的函数 solve_ivp()，method 设置为 Radau，进行数值求解常微分方程。代码见附录 B.2

图 6是求解结果：

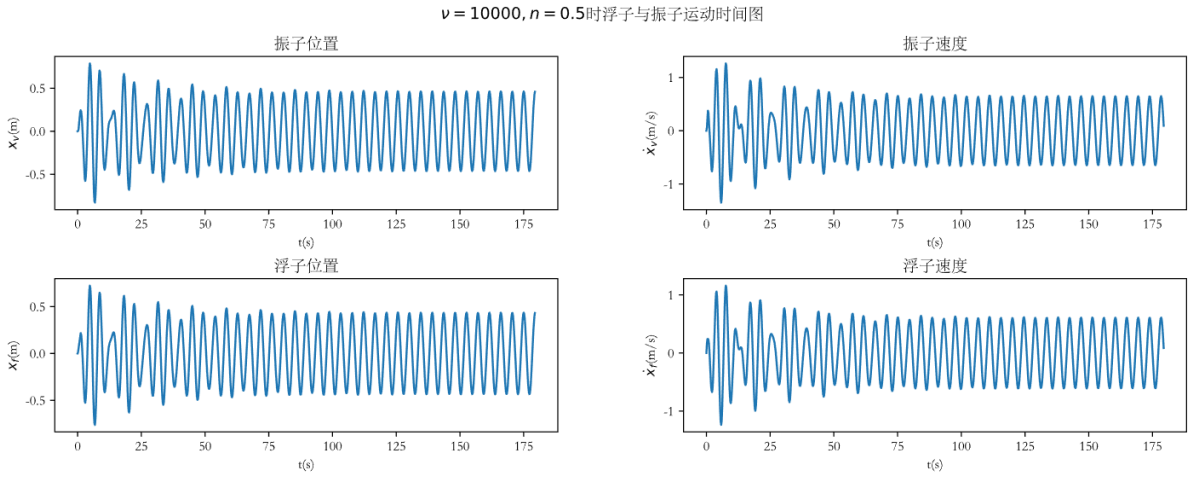

图 6: 振子与浮子运动图

我们发现这个结果和幂为 0 时差异不大

垂荡位移和速度					
	10s	20s	40s	60s	100s
浮子位移	-0.2059	-0.6111	0.2688	-0.3272	-0.08841
浮子速度	0.6528	-0.2548	0.2953	-0.4915	-0.6098
振子位移	-0.2346	-0.6611	0.2802	-0.3496	-0.09349
振子速度	-0.6999	-0.2770	0.3125	-0.5256	-0.6501

表 5: 垂荡位移速度表

6.2 问题二的求解

6.2.1 第一小问

由于浮子在波浪中长时间运动进行能量转换, 因此我们只需考虑稳定状态下的输出功率。将方程扩展至复数域, 设稳定后 x_v,x_f 的复振幅为 $\widetilde{A}_v$,$\widetilde{A}_f$

$$x_v = \widetilde{A}_v e^{-i\omega t}$$

$$x_f = \widetilde{A}_f e^{-i\omega t}$$

带入式 (3) 中的方程可得

$$-\omega^2 m_v \widetilde{A}_v = -k(\widetilde{A}_v - \widetilde{A}_f) + i\omega\nu(\widetilde{A}_v - \widetilde{A}_f)$$

$$-\omega^2(m_f + m_a)\widetilde{A}_f = k(\widetilde{A}_v - \widetilde{A}_f) - i\omega\nu(\widetilde{A}_v - \widetilde{A}_f) + i\omega\mu\widetilde{A}_f - r\widetilde{A}_f + f$$

解得复振幅

$$\widetilde{A}_v = \frac{f(-k + i\nu\omega)}{(-k + i\nu\omega)^2 - (k - m_v\omega^2 - i\nu\omega)\left(k - (\omega^2(m_a + m_f)) + r - i\mu\omega - i\nu\omega\right)}$$

$$\widetilde{A}_f = \frac{k - \omega^2 m_v - i\omega\nu}{k - i\omega\nu}\widetilde{A}_v$$

代入数值即可得到 $\widetilde{A}_v$,$\widetilde{A}_v$ 关于 ν 的表达式平均功率

$$\begin{aligned}\bar{P} &= \frac{1}{T}\int_0^T \nu(\dot{x}_v - \dot{x}_f)^2 dt \\ &= \frac{1}{T}\int_0^T \nu\left(\frac{\partial(\Re(x_f)\cos(wt) + \Im(x_f)\sin(wt))}{\partial t} - \frac{\partial(\Re(x_v)\cos(wt) + \Im(x_v)\sin(wt))}{\partial t}\right)^2\end{aligned}$$

其中周期 $T = \frac{2\pi}{\omega}$ 我们将 $\bar{P}$ 对 ν 求导即可寻找到 $\nu \in [0, 10000]$ 的极值点

经计算 $\nu = 37193.8$ 时平均功率最大, 相关计算过程见附件 B.1.3

6.2.2 第二小问

由于缺少解析式，无法进行求导，因此我们只能通过数值方法寻求最优。

瞬时功率的计算比较简单，只需要将微分方程中结果中每个时刻的 $\dot{x}_v, \dot{x}_f$ 项相减、平方再乘以系数即可。代码见附录 B.2。

求平均功率需要用到积分。由于结果是离散时刻的函数值，因此我们采用复合辛普森方法进行积分，再进行简单的计算得到平均功率。代码见附录 B.2。

最后是寻找最优的输出功率。首先我们通过作图来大致把握平均功率与参数之间的关系，得到的图像见图 7和图 8:

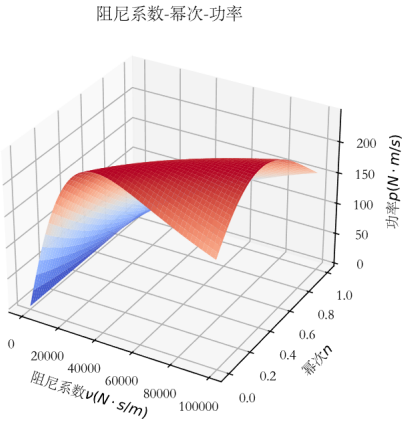

图 7: 功率曲面图

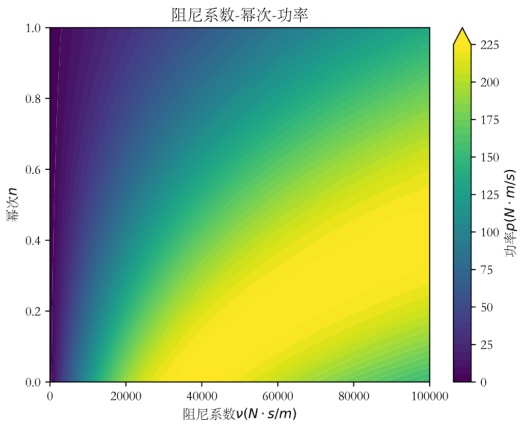

图 8: 功率色度图

从图中可以看出函数是非凹的，因此局部最优即全局最优。考虑到计算速度，我们应当用局部最优查找方法，所以最终我们选择了 python 的 scipy 库中现成的非常常用的局部最优优化方法——下山单纯形法。优化过程如图 9。

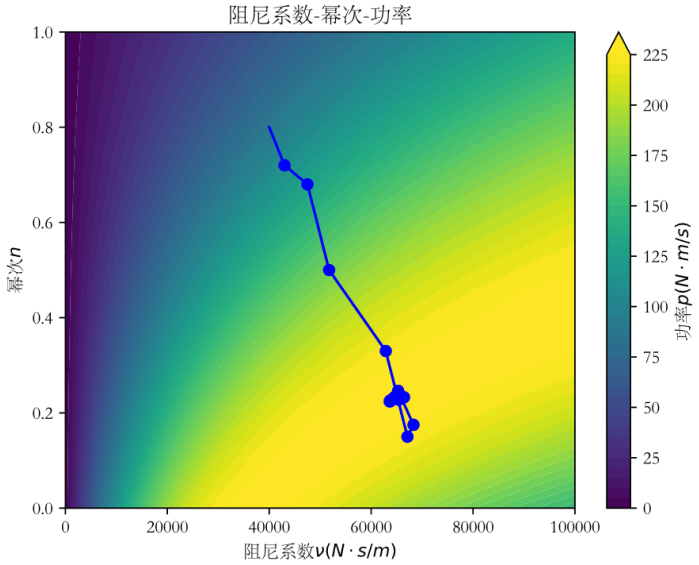

图 9: 优化过程

最终我们在 $\nu \approx 63414, n \approx 0.2259$ 处寻得最小值。此状态下的运动图像如图 10所示。

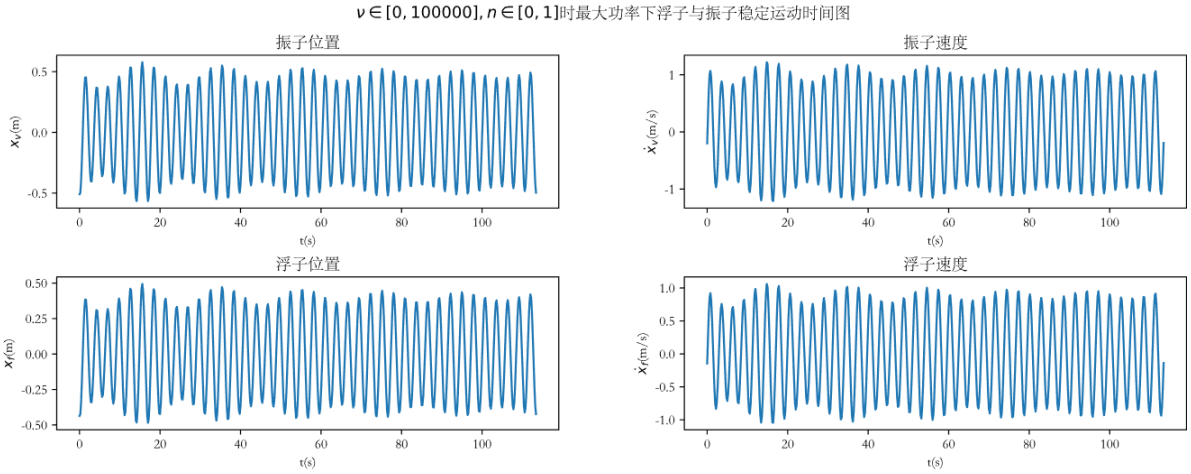

图 10: 最优功率

6.3 问题三的求解

问题三求解的方程为式 (10)

由于方程中含有非线性项（三角函数项），因此要得到解析解是一件十分困难的事，我们不得不通过数值解来求解这个非线性常微分方程。

如问题一的第二小问，我们在分析过后认为方程刚性略大，因此采用了 IRK 方法求解微分方程，最终结果如图 11。代码见附录 B.3。

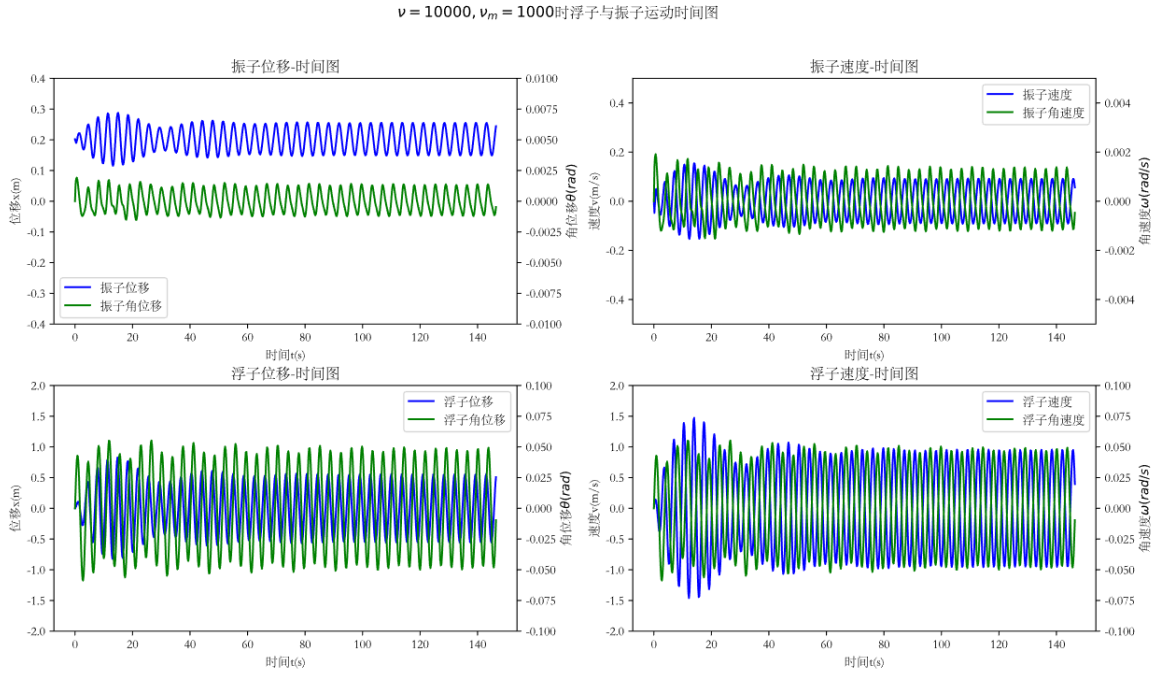

图 11: 浮子与振子的运动图

从上图我们可以看出浮子相对转动很小。

表 6为 10s、20s、40s、60s 和 100s 时刻的垂荡、纵摇位移和速度:

垂荡纵摇位移和速度					
	10s	20s	40s	60s	100s
浮子位移	-0.5266	-0.70411	0.36959	-0.31979	-0.05042
浮子速度	0.96987	-0.26961	0.75612	-0.72131	-0.94539
浮子角位移	-0.01009	0.00578	-0.03005	0.0242	0.00723
浮子角速度	-0.04063	0.00626	-0.01655	0.0304	0.04798
振子位移	0.13171	0.13457	0.22545	0.18139	0.20945
振子速度	0.06849	-0.04966	0.08751	-0.07711	-0.0897
振子角位移	-0.00012	0.00015	-0.00076	0.0005	0.00018
振子角速度	-0.00072	0.00008	-0.00075	0.00045	0.00108

表 6: 垂荡纵摇位移速度表

6.4 问题四的求解

同问题二的第二小问相仿，我们先通过作图对题目进行一个大致地把握，如图 12和图 13。

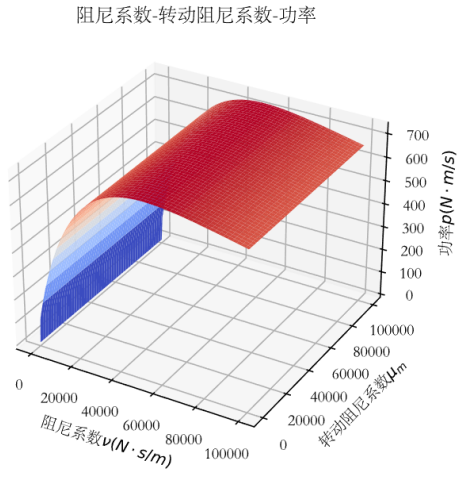

图 12: 功率曲面图

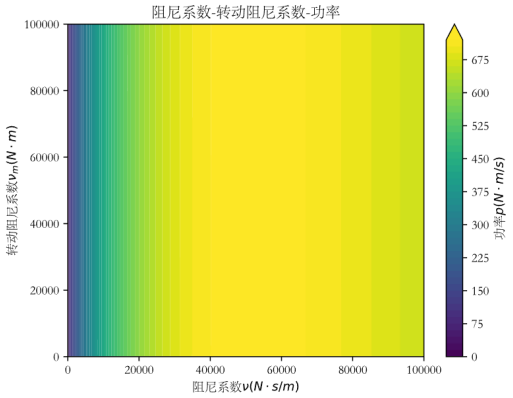

图 13: 功率色度图

在图中我们可以看出转动阻尼系数对总功率影响很小。这是由于浮子转动相对于浮子振动来说过小（从图 11可以看出小了约两个数量级），同时转动阻尼数量级相同，导致转动阻尼产生的功率会比振动阻尼产生的功率小四个数量级。

而后我们采用数值的方法对功率进行优化，优化过程如图 14。

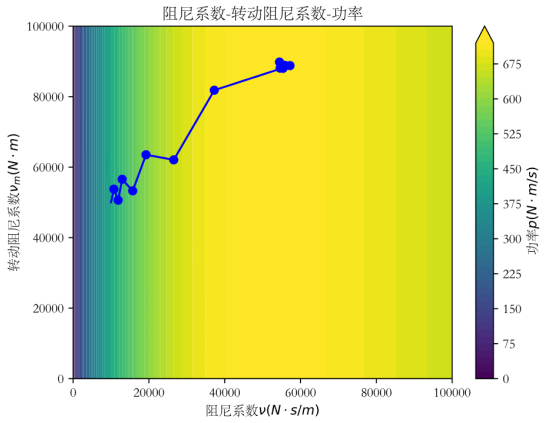

图 14: 单纯形法优化过程

最终在 $\nu = 55384.971135991254, \nu_m = 88032.31963484568$ 处我们取得平均功率最大值为 389.74，其稳定运动图像如图 15。代码见附录 B.3。

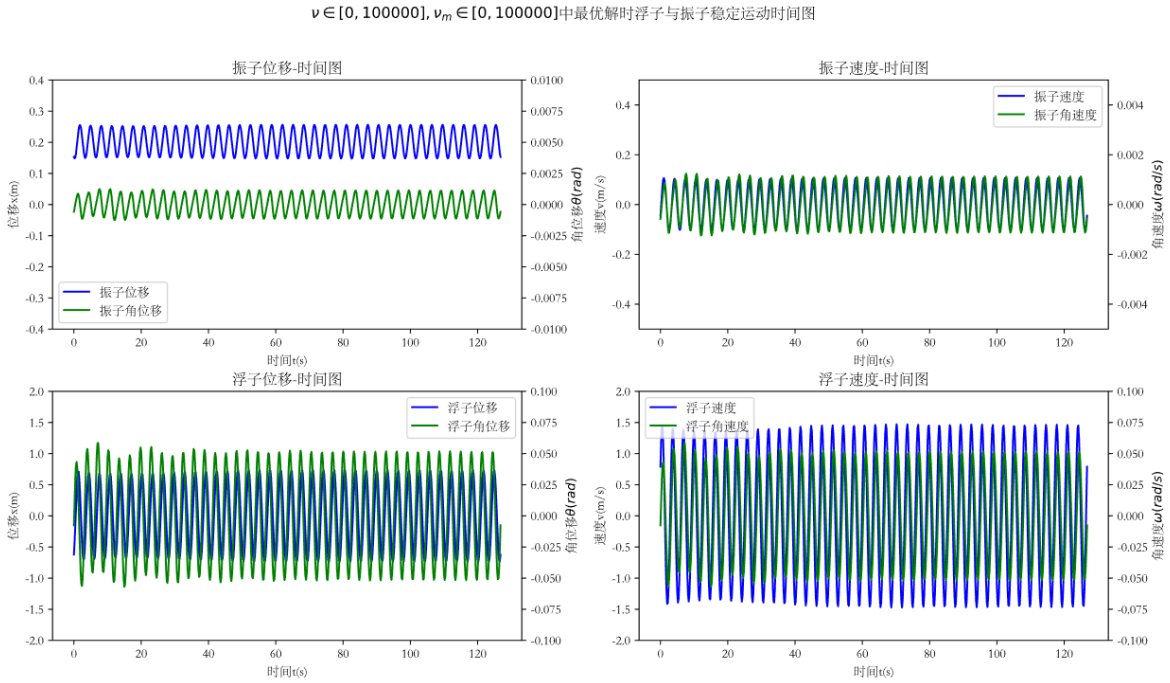

图 15: 稳定运动图像

七、模型评价

7.1 精度分析

整个数值求解的过程中，误差主要来自于辛普森法数值积分、RK 迭代以及下山单纯形法优化。优化的误差我们设置成了 10^{-5}，因此可以暂时先忽略。精度主要由辛普森积分的精度和 RK 迭代精度共同制约。

首先是辛普森积分的精度。可以证明，复合辛普森迭代的精度为四阶精度（$E_n^s(f) = -\frac{b-a}{180}h^4 f^{(4)}(\xi)$）[3].

其次是 RK 方法的精度。从文献中我们查得经典 4 阶 RK 方法的单步误差在 $O(h^5)$[4]，而步数 s 反比于步长，不难计算总体误差的量级：

$$E_t = (1+E)^s - 1 \approx sE = O(h^4)$$

因此它也是四阶精度的。

综上，我们的数值求解精度总体在四阶精度。由于题目规定步长为 0.2s，因此我们数值求解精度在 0.1% 的量级。

不过，在计算过程中我们注意到，功率曲面在某条曲线上的梯度极小（几乎为零），导致在误差范围内的最优值实际上不是一个点，而是一条曲线的狭长范围，在范围里的参数均可以视作是最优功率参数。对此问题的进一步分析我们放在下面的敏感度分析当中。

7.2 敏感度分析

7.2.1 激励力振幅对浮子振幅的影响

当激励力过大时，会导致位移的振幅和角位移的振幅过大时，水将没过浮子顶端边缘，导致浮子顶部进水，从而导致前面的分析失效。

我们先经过简单推导（设出当水面刚好没过时 θ_v 便可求得此时 $x_f \leq \cos\theta(1 - \tan\theta_v)$）得到水为了防止没过浮子顶端边缘，浮子的位移 x_f 和浮子角位移 θ_f 应满足

$$\frac{x_f}{\cos\theta_f} + \tan\theta_f \leq 1 \tag{15}$$

将式 15左边作为振幅指数 T_f，我们通过 python 画出了激励力振幅、激励力矩振幅与 T_f 的大致图像（其它参数均取题目参数），第二题如图 16，第三题如图 17和图 18。

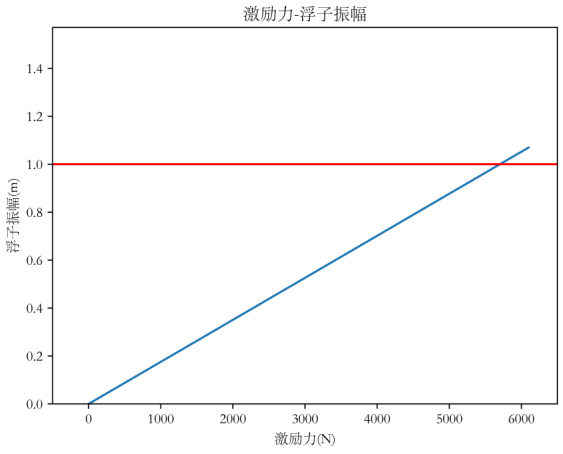

图 16: 问题二-振幅指数与垂荡激励振幅的关系

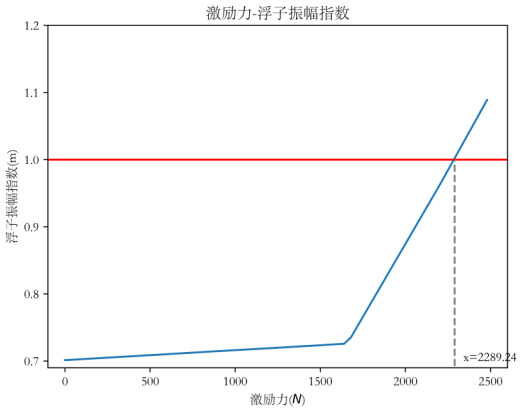

图 17: 问题四-振幅指数与垂荡激励振幅的关系

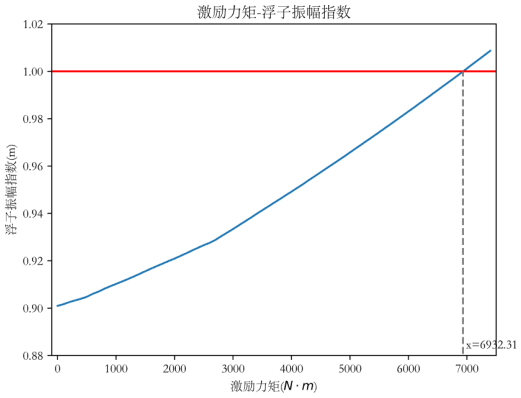

图 18: 问题四-振幅指数与纵摇激励振幅的关系

可见问题二中，当激励力振幅超过 5500 时推导可能失效；而在问题四中，当激励力振幅超过 2289.24 或者激励力矩振幅超过 6932.31 时，推导可能失效。

7.2.2 最优参数范围

我们通过网格覆盖参数可取区间，并计算每个格点上的功率大小，确定了大致的最优参数范围，如图 19和 20中亮处黑线所标出区域。

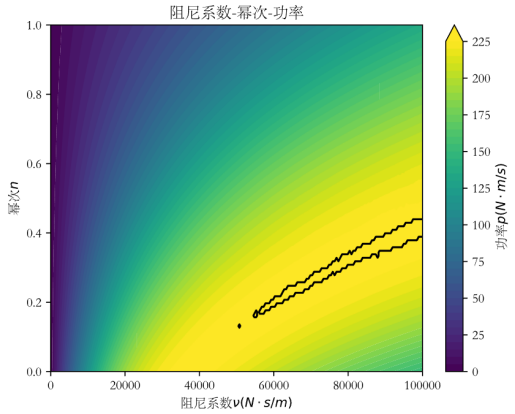

图 19: 问题二第二小问误差范围下的最优区间

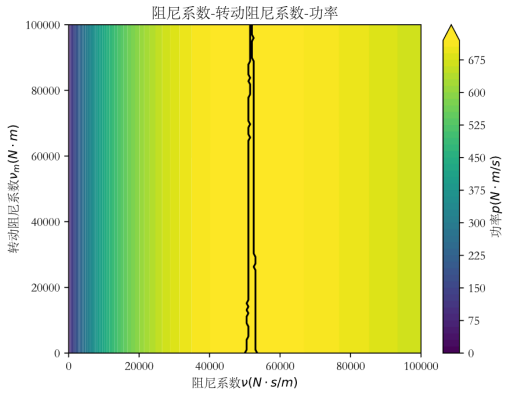

图 20: 问题三误差范围下的最优区间

在允许误差下，这些范围中的参数均可以看做是最优解，其平均功率几乎相等，这就给了阻尼器一定的选择空间。

7.3 优点

对于线性方程求出的解析解使得我们可以利用极少的计算机资源得到极高的精度。

四阶精度下的数值计算 Runge-Kutta 隐式算法考虑了方程的刚性问题，使得收敛问题不必考虑，计算资源得以节省；而 Simpson 积分算法则充分利用了迭代的结果，并且保持了四阶的精度，使得我们能够利用较少的计算机资源得到相当不错的计算模拟精度，为后续的计算留下空间。

7.4 应用、推广和改进

其它应用此模型的分析方法不仅能够分析 PTO 对波浪能的利用，也适合被用来简单地分析所有振子-浮子系统组合形成的水面设施，具有很好的适用性。

另外两个自由度：此模型分析了一个平面内的四个自由度，而利用同样的方法可以分析整个三维空间的所有六个自由度（包括另一方向的浮子与振子转动），以更好地利用海洋不同方向的波浪能。

水面高度的改进：在遇到较大的波浪时，假设 2.1就不成立了；此时可以加入对水面高度变化的变量来抵消浮子的位移和角位移。这是一个线性修正，因此，上面模型的所有分析方法仍然适用。若水面曲率较大，则只能通过数值方法进行模拟。

参考文献

[1] 朗道. 理论物理学教程 [M]. 第一卷, 力学. 北京: 高等教育出版社, 2007: P60-95

[2] 梁昆淼. 数学物理方法 [M]. 第 5 版. 北京: 高等教育出版社, 2020

[3] Weisstein E W. Simpson's rule[J]. https://mathworld. wolfram. com/, 2003.

[4] L. FOX, CHAPTER 2 - METHODS OF RUNGE—KUTTA TYPE, Editor(s): L. FOX, Numerical Solution of Ordinary and Partial Differential Equations, Pergamon, 1962, Pages 16-27, ISBN 9780080096605

[5] 束芳芳, 蔡峰, 廖康明. 福建沿海近岸海域频谱分析 [C]//. 第十六届中国海洋（岸）工程学术讨论会论文集（上册）.,2013:482-489.

八、附录

A 文件列表

支撑材料

1. support
    - analysis.py
    - calcdiff12.py
    - calcdiff34.py
    - Jacobian.nb
    - Laplace.nb
    - optimize12.py
    - optimize34.py
    - problem12.ipynb
    - problem34.ipynb
    - simpson.py
    - 复振幅求解.nb
2. results
    - result1-1.xlsx
    - result1-2.xlsx
    - result3.xlsx

B 代码

B.1 Mathematica 代码

B.1.1 Laplace 逆变换的代码 (Laplace.nb)

In[]:=
```
Solve[
  解方程
  {mv p^2 xv == (-k - v p) (xv - xf),
   (mf + ma) p^2 xf == (k + v p) (xv - xf) - u p xf - r xf + f * p/(p^2 + w^2)},
  {xf, xv}]
```

Out[]=
$$\left\{\left\{xf \to \frac{f\,p\,\left(k+mv\,p^2+p\,v\right)}{\left(-(-k-p\,v)^2+\left(k+mv\,p^2+p\,v\right)\left(k+(ma+mf)\,p^2+r+p\,u+p\,v\right)\right)\left(p^2+w^2\right)},\right.\right.$$
$$\left.\left.xv \to (f\,p\,(k+p\,v))\Big/\left(\left(k\,ma\,p^2+k\,mf\,p^2+k\,mv\,p^2+ma\,mv\,p^4+mf\,mv\,p^4+k\,r+mv\,p^2\,r+k\,p\,u+mv\,p^3\,u+ma\,p^3\,v+mf\,p^3\,v+mv\,p^3\,v+p\,r\,v+p^2\,u\,v\right)\left(p^2+w^2\right)\right)\right\}\right\}$$

In[]:=
```
(*不变参数*)
mv = 2433;
mf = 4866;
k = 80 000;
g = 9.8;
r = 1025 * g * Pi;
    圆周率
v = 10 000
km = 250 000;
rm = 8890.7;
l = 0.5;
w = 1.4005;
ma = 1335.535;
μ = u = 656.3616;
(*μm=um=151.4388;*)
f = 6250;
```

Out[]=
```
10 000
```

In[]:=
```
xf = f p (k + mv p^2 + p v) / ((-(-k - p v)^2 + (k + mv p^2 + p v) (k + (ma + mf) p^2 + r + p u + p v)) (p^2 + w^2))
xv = (f p (k + p v)) /
   ((k ma p^2 + k mf p^2 + k mv p^2 + ma mv p^4 + mf mv p^4 + k r + mv p^2 r + k p u + mv p^3 u +
       ma p^3 v + mf p^3 v + mv p^3 v + p r v + p^2 u v) (p^2 + w^2))
```

Out[]=
```
(6250 p (80 000 + 10 000 p + 2433 p^2)) / ((1.9614 + p^2) (-(-80 000 - 10 000 p)^2 +
      (80 000 + 10 000 p + 2433 p^2) × (111 557. + 10 656.4 p + 6201.54 p^2)))
```

Out[]=
```
(6250 p (80 000 + 10 000 p)) / ((1.9614 + p^2) ×
     (2.52458 × 10^9 + 3.68082 × 10^8 p + 7.74105 × 10^8 p^2 + 8.79423 × 10^7 p^3 + 1.50883 × 10^7 p^4))
```

浮子位移

```
Re[InverseLaplaceTransform[xf, p, t]] /. t → {Range[0, 40 × 2 Pi / w, 0.2]}
```
拉普拉斯反变换 范围 圆周率

浮子速度

In[ ]:=
```
Re[D[InverseLaplaceTransform[xv, p, t], t]] /. t → {Range[0, 40 × 2 Pi / w, 0.2]}
```
拉普拉斯反变换 范围 圆周率

振子位移

```
Re[InverseLaplaceTransform[xv, p, t]] /. t → {Range[0, 40 × 2 Pi / w, 0.2]}
```
拉普拉斯反变换 范围 圆周率

振子速度

```
Re[D[InverseLaplaceTransform[xf, p, t], t]] /. t → {Range[0, 40 × 2 Pi / w, 0.2]}
```
拉普拉斯反变换 范围 圆周率

画图像

In[ ]:=
```
Xf = Re[InverseLaplaceTransform[xf, p, t]];
```
拉普拉斯反变换
```
Xv = Re[InverseLaplaceTransform[xv, p, t]];
```
拉普拉斯反变换
```
Plot[ {Xv, Xf}, {t, 0, 40 * 2 Pi / w}]
```
绘图 圆周率
```
(*Plot[{D[Xf,t]},{t,0,200}]*)
```
绘图 偏导

B.1.2 通过 Jacobian 矩阵估计方程刚性的代码 (Jacobian.nb)

```
In[•]:= (*数据*)
mv = 2433;
mf = 4866;
k = 80 000;
g = 9.8;
r = 1025 * g * Pi;
           圆周率
w = 1.4005;
ma = 1335.535;
μ = u = 656.3616;
f = 6250;
v = 10 000;

In[•]:= (*xx1'=(-k x1+k x2- v (xx1-xx2)Sqrt[Abs[xx1[t]-xx2[t]]])/mv;
                                    平方根 绝对值
xx2'=(-k x2+k x1+v (xx1-xx2)Sqrt[Abs[xx1[t]-xx2[t]]]-μ xx2+f Cos[w t]-r x2)/
                                  平方根 绝对值                  余弦
  (mf+ma);
x1'=xx1;
x2'=xx2;*)
jacobian =
 N[D[{xx1, xx2, (-k x1 + k x2 - v (xx1 - xx2) Sqrt[xx1 - xx2]) / mv, (-k x2 + k x1 +
 偏导                                    平方根
      v (xx1 - xx2) Sqrt[xx1 - xx2] - μ xx2 - r x2) / (mf + ma)}, {{x1, x2, xx1, xx2}}]]
                    平方根

Out[•]= {{0., 0., 1., 0.}, {0., 0., 0., 1.},
  {-32.8812, 32.8812, -6.16523 √(xx1 - 1. xx2), 6.16523 √(xx1 - 1. xx2)}, {12.9,
   -17.9887, 2.41876 √(xx1 - 1. xx2), 0.00016125 × (-656.362 - 15 000. √(xx1 - 1. xx2))}}

jacobian /. {xx1 → 1, xx2 → -1} (*速度区间在正负一内,速度差最大求最大刚度*)

Out[•]= {{0., 0., 1., 0.}, {0., 0., 0., 1.},
  {-32.8812, 32.8812, -8.71895, 8.71895}, {12.9, -17.9887, 3.42064, -3.52648}}
```

B.1.3 求复振幅以及最优功率解析解 (复振幅求解.nb)

```
In[]:= s = Solve[{-w^2 mv Av == -k Av + k Af - v (-I w Av + I w Af),
          -w^2 (mf + ma) Af == -k Af + k Av + v (-I w Av + I w Af) + I u w Af - r Af + f}, {Av, Af}]
```

```
Out[]= {{Av → f (-k + i v w) / ((-k + i v w)^2 - (k + r - i u w - i v w - (ma + mf) w^2) (k - i v w - mv w^2)),
  Af → -((f (k - i v w - mv w^2)) / (-k r + i k u w + i r v w + k ma w^2 + k mf w^2 + k mv w^2 + mv r w^2 +
      u v w^2 - i mv u w^3 - i ma v w^3 - i mf v w^3 - i mv v w^3 - ma mv w^4 - mf mv w^4))}}
```

```
In[]:= av = f (-k + i v w) / ((-k + i v w)^2 - (k + r - i u w - i v w - (ma + mf) w^2) (k - i v w - mv w^2));
af = -((f (k - i v w - mv w^2)) / (-k r + i k u w + i r v w + k ma w^2 + k mf w^2 + k mv w^2 + mv r w^2 +
      u v w^2 - i mv u w^3 - i ma v w^3 - i mf v w^3 - i mv v w^3 - ma mv w^4 - mf mv w^4));
```

```
In[]:= mv = 2433;
mf = 4866;
k = 80 000;
g = 9.8;
r = 1025 * g * Pi;
w = 2.2143;
ma = 1165.992;
μ = u = 167.8395;
f = 4890;
```

```
In[]:= p = Integrate[v 1 / (2 Pi / w) (D[Re[af] Cos[w t] + Im[af] Sin[w t], t] -
          D[Re[av] Cos[w t] + Im[av] Sin[w t], t])^2, {t, 0, 2 Pi / w}];
```

```
In[]:= p1 = Together[ComplexExpand[p]];
```

```
In[]:= Solve[D[p1, v] == 0 && D[p1, {v, 2}] < 0 && v > 0, v, Reals]
```

```
Out[]= {{v → 37 193.8}}
```

In[]:=
```
s = Solve[{-w^2 mv Av == -k Av + k Af - v (-I w Av + I w Af),
  -w^2 (mf + ma) Af == -k Af + k Av + v (-I w Av + I w Af) + I u w Af - r Af + f}, {Av, Af}]
```

Out[]=
$$\left\{\left\{Av \to \frac{f\,(-k+i\,v\,w)}{(-k+i\,v\,w)^2-\left(k+r-i\,u\,w-i\,v\,w-(ma+mf)\,w^2\right)\left(k-i\,v\,w-mv\,w^2\right)},\right.\right.$$
$$\left.\left.Af \to -\left(\left(f\left(k-i\,v\,w-mv\,w^2\right)\right)\Big/\left(-k\,r+i\,k\,u\,w+i\,r\,v\,w+k\,ma\,w^2+k\,mf\,w^2+k\,mv\,w^2+mv\,r\,w^2+u\,v\,w^2-i\,mv\,u\,w^3-i\,ma\,v\,w^3-i\,mf\,v\,w^3-i\,mv\,v\,w^3-ma\,mv\,w^4-mf\,mv\,w^4\right)\right)\right\}\right\}$$

In[]:=
```
av = f (-k + i v w) / ((-k + i v w)^2 - (k + r - i u w - i v w - (ma + mf) w^2) (k - i v w - mv w^2));
af = -((f (k - i v w - mv w^2)) / (-k r + i k u w + i r v w + k ma w^2 + k mf w^2 + k mv w^2 + mv r w^2 + u v w^2 - i mv u w^3 - i ma v w^3 - i mf v w^3 - i mv v w^3 - ma mv w^4 - mf mv w^4));
```

In[]:=
```
mv = 2433;
mf = 4866;
k = 80 000;
g = 9.8;
r = 1025 * g * Pi;
w = 2.2143;
ma = 1165.992;
μ = u = 167.8395;
f = 4890;
```

In[]:=
```
p = Integrate[v 1/(2 Pi / w) (D[Re[af] Cos[w t] + Im[af] Sin[w t], t] -
    D[Re[av] Cos[w t] + Im[av] Sin[w t], t])^2, {t, 0, 2 Pi / w}];
```

In[]:=
```
p1 = Together[ComplexExpand[p]];
```

In[]:=
```
Solve[D[p1, v] == 0 && D[p1, {v, 2}] < 0 && v > 0, v, Reals]
```

Out[]=
$$\{\{v \to 37\,193.8\}\}$$

B.2 问题一、二的 python 代码

B.2.1 Simpson 积分的 python 代码 (三四题同)(simpson.py)

```
1
2
3  def Dint(arr,h):
4      #复合辛普森积分
5      n=len(arr)
6      sum=0
7      if (n%2==0):
8          sum+=(arr[-1]+arr[-2])*h/2
9          n-=1
10     tsum=arr[0]+arr[n-1]
11     for i in range(1,n-1):
12         if(i%2):
13             tsum+=4*arr[i]
14         else:
15             tsum+=2*arr[i]
16     sum+=tsum*h/3
17     return sum
```

B.2.2 微分方程计算 (calcdiff12.py)

```
1  import scipy.integrate as intode
2  import numpy as np
3  import matplotlib.pyplot as plt
4  from matplotlib import cm
5  import time
6  import scipy.optimize as opt
7  from mpl_toolkits.mplot3d import Axes3D
8  import matplotlib_inline.backend_inline
9  matplotlib_inline.backend_inline.set_matplotlib_formats('
       svg')
10 import pandas as pd
11
12 solvemethod='Radau'
13 problem=2
```

```
plt.rcParams['font.sans-serif'] = ['STSong']
plt.rcParams['axes.unicode_minus'] = False

k=80000
r=1025*9.8*np.pi
mv=2433
mf=4866
if(problem==1):
    ma=1335.535
    mu=656.3616
    w=1.4005
    f=6250
elif(problem==2):
    ma=1165.992
    mu=167.8395
    w=2.2143
    f=4890
prd=2*np.pi/w
tmax=prd*40
titv=0.2
nu1=10000
n1=0.5

start=time.time()
try:
    x0=np.array([xres1[i][-1] for i in range(4)])
    x1=np.array([xres2[i][-1] for i in range(4)])
except:
    x0=np.array([0,0,0,0])
    x1=np.array([0,0,0,0])
def equ1(t,x,nu):
    xv=x[0]
    xf=x[1]
    dxv=x[2]
    dxf=x[3]
    fxv=dxv
    fxf=dxf
```

```
51        fdxv=(-k*xv+k*xf-nu*(dxv-dxf))/mv
52        m=mf+ma
53        fdxf=(-k*xf+k*xv+nu*(dxv-dxf)-mu*dxf+f*np.cos(w*t)-r*xf
              )/m
54        return np.array([fxv,fxf,fdxv,fdxf])
55    def fun1(t,x):
56        return equ1(t,x,nu1)
57    def equ2(t,x,nu,n):
58        xv=x[0]
59        xf=x[1]
60        dxv=x[2]
61        dxf=x[3]
62        fxv=dxv
63        fxf=dxf
64        fdxv=(-k*xv+k*xf-nu*(dxv-dxf)*(np.abs(dxv-dxf))**n)/mv
65        m=mf+ma
66        fdxf=(-k*xf+k*xv+nu*(dxv-dxf)*(np.abs(dxv-dxf))**n-mu*
              dxf+f*np.cos(w*t)-r*xf)/m
67        return np.array([fxv,fxf,fdxv,fdxf])
68    def fun2(t,x):
69        return equ2(t,x,nu1,n1)
70    print(x0)
71    print(x1)
72    te=np.arange(start=0,stop=tmax,step=titv)
73    sol1 = intode.solve_ivp(fun1,(te[0],te[-1]),x0,method=
          solvemethod,t_eval=te)
74    sol2 = intode.solve_ivp(fun2,(te[0],te[-1]),x1,method=
          solvemethod,t_eval=te)
75    end=time.time()
76    print(end-start)
77
78    xres1=sol1.y
79    xres2=sol2.y
80    tres=sol1.t
81
82    data=pd.DataFrame(np.transpose(xres1))
83    writer=pd.ExcelWriter('xres1-temp.xlsx')
```

```
84  data.to_excel(writer,'page_1',float_format='%.5f')
85  writer.save()
86  writer.close()
87  data=pd.DataFrame(np.transpose(xres2))
88  writer=pd.ExcelWriter('xres2-temp.xlsx')
89  data.to_excel(writer,'page_1',float_format='%.5f')
90  writer.save()
91  writer.close()
92  data=pd.DataFrame(np.transpose(tres))
93  writer=pd.ExcelWriter('t-temp.xlsx')
94  data.to_excel(writer,'page_1',float_format='%.5f')
95  writer.save()
96  writer.close()
97
98  x1vres=xres1[0]
99  x1fres=xres1[1]
100 v1vres=xres1[2]
101 v1fres=xres1[3]
102 x2vres=xres2[0]
103 x2fres=xres2[1]
104 v2vres=xres2[2]
105 v2fres=xres2[3]
106
107 def draw1():
108     plt.figure(figsize=(15,5))
109     plt.suptitle(r'$\nu\in[0,100000],n=0$时最大功率下浮子与
        振子运动时间图')
110     plt.subplots_adjust(wspace=0.25,hspace=0.45)
111     plt.subplot(2,2,1)
112     plt.plot(tres,x1vres)
113     plt.title('振子位置')
114     plt.xlabel(r't(s)')
115     plt.ylabel(r'$x_{v}$(m)')
116     plt.subplot(2,2,2)
117     plt.plot(tres,v1vres)
118     plt.title('振子速度')
119     plt.xlabel(r't(s)')
```

```
120         plt.ylabel(r'$\dot x_{v}$(m/s)')
121         plt.subplot(2,2,3)
122         plt.plot(tres,x1fres)
123         plt.title('浮子位置')
124         plt.xlabel(r't(s)')
125         plt.ylabel(r'$x_{f}$(m)')
126         plt.subplot(2,2,4)
127         plt.plot(tres,v1fres)
128         plt.title('浮子速度')
129         plt.xlabel(r't(s)')
130         plt.ylabel(r'$\dot x_{f}$(m/s)')
131
132     def draw2():
133         plt.figure(figsize=(15,5))
134         plt.suptitle(r'$\nu\in[0,100000],n\in[0,1]$时最大功率下
                浮子与振子稳定运动时间图')
135         plt.subplots_adjust(wspace=0.25,hspace=0.45)
136         plt.subplot(2,2,1)
137         plt.plot(tres,x2vres)
138         plt.title('振子位置')
139         plt.xlabel(r't(s)')
140         plt.ylabel(r'$x_{v}$(m)')
141         plt.subplot(2,2,2)
142         plt.plot(tres,v2vres)
143         plt.title('振子速度')
144         plt.xlabel(r't(s)')
145         plt.ylabel(r'$\dot x_{v}$(m/s)')
146         plt.subplot(2,2,3)
147         plt.plot(tres,x2fres)
148         plt.title('浮子位置')
149         plt.xlabel(r't(s)')
150         plt.ylabel(r'$x_{f}$(m)')
151         plt.subplot(2,2,4)
152         plt.plot(tres,v2fres)
153         plt.title('浮子速度')
154         plt.xlabel(r't(s)')
155         plt.ylabel(r'$\dot x_{f}$(m/s)')
```

```
156
157 draw1()
158 plt.show()
```

B.2.3 优化 (optimize12.py)

```
1  from simpson import *
2  import scipy.integrate as intode
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from matplotlib import cm
6  import time
7  import scipy.optimize as opt
8  from mpl_toolkits.mplot3d import Axes3D
9  import matplotlib_inline.backend_inline
10 matplotlib_inline.backend_inline.set_matplotlib_formats('
       svg')
11 import pandas as pd
12
13 solvemethod='Radau'
14 problem=2
15 plt.rcParams['font.sans-serif'] = ['STSong']
16 plt.rcParams['axes.unicode_minus'] = False
17 k=80000
18 r=1025*9.8*np.pi
19 mv=2433
20 mf=4866
21 if(problem==1):
22     ma=1335.535
23     mu=656.3616
24     w=1.4005
25     f=6250
26 elif(problem==2):
27     ma=1165.992
28     mu=167.8395
29     w=2.2143
30     f=4890
```

```
prd=2*np.pi/w
tmax=prd*40
titv=0.2
nu1=10000
n1=0.5

try:
    x0=np.array([xres1[i][-1] for i in range(4)])
    x1=np.array([xres2[i][-1] for i in range(4)])
except:
    x0=np.array([0,0,0,0])
    x1=np.array([0,0,0,0])

def equ1(t,x,nu):
    xv=x[0]
    xf=x[1]
    dxv=x[2]
    dxf=x[3]
    fxv=dxv
    fxf=dxf
    fdxv=(-k*xv+k*xf-nu*(dxv-dxf))/mv
    m=mf+ma
    fdxf=(-k*xf+k*xv+nu*(dxv-dxf)-mu*dxf+f*np.cos(w*t)-r*xf
        )/m
    return np.array([fxv,fxf,fdxv,fdxf])
def fun1(t,x):
    return equ1(t,x,nu1)
def equ2(t,x,nu,n):
    xv=x[0]
    xf=x[1]
    dxv=x[2]
    dxf=x[3]
    fxv=dxv
    fxf=dxf
    fdxv=(-k*xv+k*xf-nu*(dxv-dxf)*(np.abs(dxv-dxf))**n)/mv
    m=mf+ma
    fdxf=(-k*xf+k*xv+nu*(dxv-dxf)*(np.abs(dxv-dxf))**n-mu*
```

```
            dxf+f*np.cos(w*t)-r*xf)/m
67      return np.array([fxv,fxf,fdxv,fdxf])
68  def fun2(t,x):
69      return equ2(t,x,nu1,n1)
70
71  def errequ2(nu,n):
72      sol=intode.solve_ivp(lambda t,x:equ2(t,x,nu,n),(te[0],
            te[-1]),x0,method=solvemethod,t_eval=te)
73      xres=sol.y
74      dxv1res=xres[2]
75      dxf1res=xres[3]
76      p1=nu*(dxv1res-dxf1res)**2*np.abs(dxv1res-dxf1res)**n
77      p1average=Dint(p1,titv)/tmax
78      print(nu,'&&',n,'->',p1average)
79      return -p1average
80  def calcP(nu,n,xres):
81      dxv1res=xres[2]
82      dxf1res=xres[3]
83      p1=nu*(dxv1res-dxf1res)**2*np.abs(dxv1res-dxf1res)**n
84      return Dint(p1,titv)/tmax
85
86  aa=np.linspace(start=0,stop=100000,num=200)
87  cc=np.linspace(start=0,stop=1,num=100)
88  X,Y=np.meshgrid(aa,cc)
89  dd=[[-errequ2(i,j) for j in cc] for i in aa]
90
91  Z=np.array(dd)
92  Z=np.transpose(Z)
93  fig=plt.figure()
94  ax = fig.add_subplot(111,projection='3d')
95  ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=0,
        antialiased=False)
96  plt.title('阻尼系数-幂次-功率')
97  plt.xlabel(r'阻尼系数$\nu(N\cdot␣s/m)$')
98  plt.ylabel(r'幂次$n$')
99  ax.set_zlabel(r'功率$p(N\cdot␣m/s)$')
100
```

```
cs=plt.contourf(X,Y,Z,50,extend='max')
plt.title('阻尼系数-幂次-功率')
plt.xlabel(r'阻尼系数$\nu(N\cdot s/m)$')
plt.ylabel(r'幂次$n$')
plt.colorbar(cs,label=r'功率$p(N\cdot m/s)$')
plt.contour(X,Y,Z>228.2,1,colors=['None','Black'])

nun0=[40000,0.8]
ptstore=np.array(nun0)
def drawpt(x):
    global ptstore
    plt.scatter(x[0],x[1],color='b')
    plt.plot([x[0],ptstore[0]],[x[1],ptstore[1]],color='b')
    ptstore=x
cs=plt.contourf(X,Y,Z,50,extend='max')

plt.title('阻尼系数-幂次-功率')
plt.xlabel(r'阻尼系数$\nu(N\cdot s/m)$')
plt.ylabel(r'幂次$n$')
plt.colorbar(cs,label=r'功率$p(N\cdot m/s)$')

res2=opt.minimize(lambda x:errequ2(x[0],x[1]),nun0,method='
    Nelder-Mead',bounds=[(0,100000),(0,1)],tol=0.0001,
    callback=drawpt)

print(res2)
print(-errequ2(res2.x[0],res2.x[1]))
```

B.3 问题三、四的 python 代码

B.3.1 微分方程计算 (calcdiff34.py)

```
import scipy.integrate as intode
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
```

```
import scipy.optimize as opt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib_inline.backend_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('svg')
import pandas as pd

#参数
solvemethod='Radau'
problem=4
k=80000
km=250000
r=1025*9.8*np.pi
rm=8890.7
mv=2433
mf=4866
if(problem==3):
    ma=1028.876
    Ia=7001.914
    mu=683.4558
    muM=654.3383
    w=1.7152
    f=3640
    L=1690
elif(problem==4):
    ma=1091.099
    Ia=7142.493
    mu=528.5018
    muM=1655.909
    w=1.9806
    f=1760
    L=2140
elif(problem==5):
    ma=1091.099
    Ia=7142.493
    mu=528.5018
    muM=1655.909
```

```
42          w=1.9806
43          f=1760
44          L=0
45  prd=2*np.pi/w
46  tmax=prd*40
47  titv=0.2
48  nu1=10000
49  num1=1000
50  g=9.8
51  l=0.5
52
53  #计算转动惯量
54  R=1
55  Hcone=0.8
56  Hclnd=3
57  M=mf
58  LL=np.sqrt(R**2+Hcone**2)
59  temp=np.pi*R*LL+2*np.pi*R*Hclnd
60  mcone=np.pi*R*LL*M/temp
61  mclnd=2*np.pi*R*Hclnd*M/temp
62  If=mclnd*R*R/2+mclnd*Hclnd**2/3+mcone*R*R/4+mcone*Hcone*
        Hcone/6
63  rr,h=0.5,0.5
64  Iv0=mv*rr**2/4+mv*h**2/12
65
66  #微分方程
67  def equ(t,x,nu,nuM):
68          xv=x[0]
69          xf=x[1]
70          tv=x[2]
71          tf=x[3]
72          vv=x[4]
73          vf=x[5]
74          wv=x[6]
75          wf=x[7]
76          dxv=vv
77          dxf=vf
```

```
78        dtv=wv
79        dtf=wf
80        dwf=(-rm*tf-muM*wf+nuM*wv+km*tv+L*np.cos(w*t))/(If+Ia)
81        Iv=Iv0+mv*xv**2
82        dwv=(-nuM*wv-km*tv+mv*g*xv*np.sin(tv+tf))/Iv-dwf
83        m=mf+ma+mv*np.sin(tv+tf)**2
84        dvf=(k*(xv-l+mv*g/k)*np.cos(tv+tf)+nu*vv*np.cos(tv+tf)-
              mu*vf+f*np.cos(w*t)-r*xf+np.sin(tv+tf)*(mv*(xv*dwv
              +2*vv*wv+xv*dwf+2*vv*wf)-mv*g*np.sin(tv+tf)))/m
85        dvv=(-k*(xv-l)-nu*vv-mv*g*np.cos(tv+tf)-mv*(-xv*wv**2+
              dvf*np.cos(tv+tf)-xv*wf**2-2*xv*wf*wv))/mv
86        return np.array([dxv,dxf,dtv,dtf,dvv,dvf,dwv,dwf])
87
88    #求解
89    te=np.arange(start=0,stop=tmax,step=titv)
90    try:
91        x0=np.array([xres[i][-1] for i in range(8)])
92    except:
93        x0=np.array([l-mv*g/k,0,0,0,0,0,0,0])
94    print(x0)
95    sol=intode.solve_ivp(lambda t,x:equ(t,x,nu1,num1),(te[0],te
          [-1]),x0,method=solvemethod,t_eval=te)
96    xres=sol.y
97    tres=sol.t
98
99    #输出结果
100   data=pd.DataFrame(np.transpose(xres))
101   writer=pd.ExcelWriter('3-xres.xlsx')
102   data.to_excel(writer,'page_1',float_format='%.5f')
103   writer.save()
104   writer.close()
105   data=pd.DataFrame(np.transpose(tres))
106   writer=pd.ExcelWriter('3-t.xlsx')
107   data.to_excel(writer,'page_1',float_format='%.5f')
108   writer.save()
109   writer.close()
110   xvres=xres[0]
```

```
111 xfres=xres[1]
112 tvres=xres[2]
113 tfres=xres[3]
114 vvres=xres[4]
115 vfres=xres[5]
116 wvres=xres[6]
117 wfres=xres[7]
118 plt.rcParams['font.sans-serif'] = ['STSong']
119 plt.rcParams['axes.unicode_minus'] = False
120 fig = plt.figure(figsize=(15, 8))
121 plt.suptitle(r'$\nu\in[0,100000],\mu_m\in[0,100000]$中最优
      解时浮子与振子稳定运动时间图')
122 plt.subplots_adjust(wspace=0.25,hspace=0.25)
123 ax1 = fig.add_subplot(2,2,1)
124 ax1.set_ylim(-0.4,0.4)
125 ha11,=ax1.plot(tres,xvres,color='b')
126 ax1.set_ylabel('位移x(m)')
127 ax1.set_title("振子位移-时间图")
128 ax1.set_xlabel('时间t(s)')
129 bx1 = ax1.twinx()
130 bx1.set_ylim(-0.01,0.01)
131 hb11,=bx1.plot(tres,wvres,color='g')
132 bx1.set_ylabel(r'角位移$\theta(rad)$')
133 ax1.legend([ha11,hb11,],['振子位移','振子角位移'])
134 ax2 = fig.add_subplot(2,2,3)
135 ax2.set_ylim(-2,2)
136 ha2,=ax2.plot(tres,xfres,color='b')
137 ax2.set_ylabel('位移x(m)')
138 ax2.set_title("浮子位移-时间图")
139 ax2.set_xlabel('时间t(s)')
140 bx2 = ax2.twinx()
141 bx2.set_ylim(-0.1,0.1)
142 hb2,=bx2.plot(tres,wfres,color='g')
143 bx2.set_ylabel(r'角位移$\theta(rad)$')
144 ax2.legend([ha2,hb2],['浮子位移','浮子角位移'])
145 ax3 = fig.add_subplot(2,2,2)
146 ax3.set_ylim(-0.5,0.5)
```

```
147 ha3,=ax3.plot(tres,vvres,color='b')
148 ax3.set_ylabel('速度v(m/s)')
149 ax3.set_title("振子速度-时间图")
150 ax3.set_xlabel('时间t(s)')
151 bx3 = ax3.twinx()
152 bx3.set_ylim(-0.005,0.005)
153 hb3,=bx3.plot(tres,wvres,color='g')
154 bx3.set_ylabel(r'角速度$\omega(rad/s)$')
155 ax3.legend([ha3,hb3],['振子速度','振子角速度'])
156 ax4 = fig.add_subplot(2,2,4)
157 ax4.set_ylim(-2,2)
158 ha4,=ax4.plot(tres,vfres,color='b')
159 ax4.set_ylabel('速度v(m/s)')
160 ax4.set_title("浮子速度-时间图")
161 ax4.set_xlabel('时间t(s)')
162 bx4 = ax4.twinx()
163 bx4.set_ylim(-0.1,0.1)
164 hb4,=bx4.plot(tres,wfres,color='g')
165 bx4.set_ylabel(r'角速度$\omega(rad/s)$')
166 ax4.legend([ha4,hb4],['浮子速度','浮子角速度'])
167
168 plt.show()
```

B.3.2 优化 (optimize34.py)

```
1  from simpson import *
2  import scipy.integrate as intode
3  import numpy as np
4  import matplotlib.pyplot as plt
5  from matplotlib import cm
6  import time
7  import scipy.optimize as opt
8  from mpl_toolkits.mplot3d import Axes3D
9  import matplotlib_inline.backend_inline
10 matplotlib_inline.backend_inline.set_matplotlib_formats('
       svg')
11 import pandas as pd
```

```
12
13 #参数
14 solvemethod='Radau'
15 problem=4
16 k=80000
17 km=250000
18 r=1025*9.8*np.pi
19 rm=8890.7
20 mv=2433
21 mf=4866
22 if(problem==3):
23         ma=1028.876
24         Ia=7001.914
25         mu=683.4558
26         muM=654.3383
27         w=1.7152
28         f=3640
29         L=1690
30 elif(problem==4):
31         ma=1091.099
32         Ia=7142.493
33         mu=528.5018
34         muM=1655.909
35         w=1.9806
36         f=1760
37         L=2140
38 elif(problem==5):
39         ma=1091.099
40         Ia=7142.493
41         mu=528.5018
42         muM=1655.909
43         w=1.9806
44         f=1760
45         L=0
46 prd=2*np.pi/w
47 tmax=prd*40
48 titv=0.2
```

```
49 nu1=10000
50 num1=1000
51 g=9.8
52 l=0.5
53
54 #计算转动惯量
55 R=1
56 Hcone=0.8
57 Hclnd=3
58 M=mf
59 LL=np.sqrt(R**2+Hcone**2)
60 temp=np.pi*R*LL+2*np.pi*R*Hclnd
61 mcone=np.pi*R*LL*M/temp
62 mclnd=2*np.pi*R*Hclnd*M/temp
63 If=mclnd*R*R/2+mclnd*Hclnd**2/3+mcone*R*R/4+mcone*Hcone*
       Hcone/6
64 rr,h=0.5,0.5
65 Iv0=mv*rr**2/4+mv*h**2/12
66
67 #微分方程
68 def equ(t,x,nu,nuM):
69     xv=x[0]
70     xf=x[1]
71     tv=x[2]
72     tf=x[3]
73     vv=x[4]
74     vf=x[5]
75     wv=x[6]
76     wf=x[7]
77     dxv=vv
78     dxf=vf
79     dtv=wv
80     dtf=wf
81     dwf=(-rm*tf-muM*wf+nuM*wv+km*tv+L*np.cos(w*t))/(If+Ia)
82     Iv=Iv0+mv*xv**2
83     dwv=(-nuM*wv-km*tv+mv*g*xv*np.sin(tv+tf))/Iv-dwf
84     m=mf+ma+mv*np.sin(tv+tf)**2
```

```
85      dvf=(k*(xv-l+mv*g/k)*np.cos(tv+tf)+nu*vv*np.cos(tv+tf)-
            mu*vf+f*np.cos(w*t)-r*xf+np.sin(tv+tf)*(mv*(xv*dwv
            +2*vv*wv+xv*dwf+2*vv*wf)-mv*g*np.sin(tv+tf)))/m
86      dvv=(-k*(xv-l)-nu*vv-mv*g*np.cos(tv+tf)-mv*(-xv*wv**2+
            dvf*np.cos(tv+tf)-xv*wf**2-2*xv*wf*wv))/mv
87      return np.array([dxv,dxf,dtv,dtf,dvv,dvf,dwv,dwf])
88
89  #求解
90  te=np.arange(start=0,stop=tmax,step=titv)
91  try:
92      x0=np.array([xres[i][-1] for i in range(8)])
93  except:
94      x0=np.array([l-mv*g/k,0,0,0,0,0,0,0])
95  print(x0)
96
97  #计算功率
98  def calcP(nu,nuM,res):
99      vv=res[4]
100     wv=res[6]
101
102     pv=nu*vv**2
103     pw=nuM*wv**2
104     p=pv+pw
105     return p
106
107 #误差函数
108 def errequ(nu,num):
109     sol=intode.solve_ivp(lambda t,x:equ(t,x,nu,num),(te[0],
            te[-1]),x0,method=solvemethod,t_eval=te)
110     xres=sol.y
111     p=calcP(nu,num,xres)
112     paverage=Dint(p,titv)/tmax
113     print(nu,'&&',num,'->',paverage)
114     return -paverage
115
116 #做初步图像
117 aa=np.linspace(start=0,stop=100000,num=200)
```

```
118 cc=np.linspace(start=0,stop=100000,num=100)
119 X,Y=np.meshgrid(aa,cc)
120 dd=[[-errequ(i,j) for j in cc] for i in aa]
121 Z=np.array(dd)
122 Z=np.transpose(Z)
123 fig=plt.figure()
124 ax = fig.add_subplot(111,projection='3d')
125 ax.plot_surface(X,Y,Z,cmap=cm.coolwarm,linewidth=0,
        antialiased=False)
126 plt.title('阻尼系数-转动阻尼系数-功率')
127 plt.xlabel(r'阻尼系数$\nu(N\cdot␣s/m)$')
128 plt.ylabel(r'转动阻尼系数$\mu_m$')
129 ax.set_zlabel(r'功率$p(N\cdot␣m/s)$')
130 cs=plt.contourf(X,Y,Z,50,extend='max')
131 plt.title('阻尼系数-转动阻尼系数-功率')
132 plt.xlabel(r'阻尼系数$\nu(N\cdot␣s/m)$')
133 plt.ylabel(r'转动阻尼系数$\nu_m(N\cdot␣m)$')
134 plt.colorbar(cs,label=r'功率$p(N\cdot␣m/s)$')
135 plt.contour(X,Y,Z>731.17,1,colors=['None','Black'])
136
137 #优化
138 nunum0=[10000,50000]
139 ptstore=np.array(nunum0)
140 def drawpt(x):
141     global ptstore
142     plt.scatter(x[0],x[1],color='b')
143     plt.plot([x[0],ptstore[0]],[x[1],ptstore[1]],color='b')
144     ptstore=x
145 cs=plt.contourf(X,Y,Z,50,extend='max')
146 plt.title('阻尼系数-转动阻尼系数-功率')
147 plt.xlabel(r'阻尼系数$\nu(N\cdot␣s/m)$')
148 plt.ylabel(r'转动阻尼系数$\nu_m(N\cdot␣m)$')
149 plt.colorbar(cs,label=r'功率$p(N\cdot␣m/s)$')
150 res=opt.minimize(lambda x:errequ(x[0],x[1]),nunum0,method='
        Nelder-Mead',bounds=[(0,100000),(0,100000)],tol=0.0001,
        callback=drawpt)
151 print(res)
```

```
print(-errequ(res.x[0],res.x[1]))
```

B.4 敏感度分析的 python 代码 (analysis.py)

```
from simpson import *
from calcdiff34 import *
import scipy.integrate as intode
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
import time
import scipy.optimize as opt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib_inline.backend_inline
matplotlib_inline.backend_inline.set_matplotlib_formats('
    svg')
import pandas as pd


from optimize34 import *

def equt(t,x,nu,nuM,f,L,w,k,km):
    xv=x[0]
    xf=x[1]
    tv=x[2]
    tf=x[3]
    vv=x[4]
    vf=x[5]
    wv=x[6]
    wf=x[7]
    dxv=vv
    dxf=vf
    dtv=wv
    dtf=wf
    dwf=(-rm*tf-muM*wf+nuM*wv+km*tv+L*np.cos(w*t))/(If+Ia)
    Iv=Iv0+mv*xv**2
    dwv=(-nuM*wv-km*tv+mv*g*xv*np.sin(tv+tf))/Iv-dwf
```

```
33        m=mf+ma+mv*np.sin(tv+tf)**2
34        dvf=(k*(xv-l+mv*g/k)*np.cos(tv+tf)+nu*vv*np.cos(tv+tf)-
              mu*vf+f*np.cos(w*t)-r*xf+np.sin(tv+tf)*(mv*(xv*dwv
              +2*vv*wv+xv*dwf+2*vv*wf)-mv*g*np.sin(tv+tf)))/m
35        dvv=(-k*(xv-l)-nu*vv-mv*g*np.cos(tv+tf)-mv*(-xv*wv**2+
              dvf*np.cos(tv+tf)-xv*wf**2-2*xv*wf*wv))/mv
36        return np.array([dxv,dxf,dtv,dtf,dvv,dvf,dwv,dwf])
37
38    def calcPow(nu,nuM,f,L,w,k,km):
39        print(x0)
40        sol=intode.solve_ivp(lambda t,x:equt(t,x,nu,nuM,f,L,w,k
              ,km),(te[0],te[-1]),x0,method=solvemethod,t_eval=te)
41        xres=sol.y
42        tres=sol.t
43        p=calcP(nu,nuM,xres)
44        index=xres[1]/np.cos(xres[3])+np.tan(xres[3])
45        return Dint(p,titv)/tmax,max(index)
46
47    nu0,num0=res.x[0],res.x[1]
48
49    ff=range(0,2500,40)
50    mm=[0 for i in ff]
51    pp=mm[:]
52    for i in range(len(ff)):
53        temp=calcPow(nu0,num0,ff[i],L,w,k,km)
54        mm[i]=temp[1]
55        pp[i]=temp[0]
56
57    plt.plot(ff,mm)
58    plt.ylim(0.69,1.2)
59    plt.xlim(-100,2600)
60    plt.plot(range(-1000,9000,1000),[1 for i in range
        (-1000,9000,1000)],color='r')
61    plt.title('激励力-浮子振幅指数')
62    plt.xlabel(r'激励力($N$)')
63    plt.ylabel('浮子振幅指数(m)')
64    xxx=2289.24
```

```
65  plt.vlines([xxx],[0],[1],linestyles='dashed',colors=['grey'
        ])
66  plt.text(xxx+50,0.7,r'x='+str(xxx))
67
68  ll=range(0,7500,100)
69 mm=[0 for i in ll]
70  pp=mm[:]
71  for i in range(len(ll)):
72      temp=calcPow(nu0,num0,f,ll[i],w,k,km)
73      mm[i]=temp[1]
74      pp[i]=temp[0]
75
76  plt.plot(ll,mm)
77  plt.ylim(0.88,1.02)
78  plt.xlim(-100,7500)
79  plt.plot(range(-1000,9000,1000),[1 for i in range
        (-1000,9000,1000)],color='r')
80  plt.title('激励力矩-浮子振幅指数')
81  plt.xlabel(r'激励力矩($N\cdot m$)')
82  plt.ylabel('浮子振幅指数(m)')
83  xxx=6932.31
84  plt.vlines([xxx],[0],[1],linestyles='dashed',colors=['grey'
        ])
85  plt.text(xxx+50,0.883,r'x='+str(xxx))
```