问题4

In [1]:

```
# TODO import

import re
import os
import sys
import hmz
import pathlib
import mitosheet
import numpy as np
import pandas as pd
import matlab.engine

import scipy
from scipy.integrate import odeint
from scipy.optimize import minimize

import time
from time import time, sleep

import copy
import random

import sympy
from sympy import limit
from sympy import diff
from sympy import integrals

import sklearn
import graphviz
from sklearn import tree
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error as MSE
from sklearn.metrics import mean_absolute_error as MAE
from sklearn.metrics import classification_report, roc_auc_score

import sko
from sko.GA import GA
```

```python
import numba
from numba import jit

import plotly
import plotly.express as px
import plotly.graph_objects as go
import plotly.figure_factory as ff
plotly.offline.init_notebook_mode()

import cufflinks as cf
cf.set_config_file(
    offline=True,
    world_readable=True,
    theme='pearl',  # cf.getThemes()
)

import matplotlib.pyplot as plt
plt.rcParams['font.sans-serif'] = ['SimHei']  # KaiTi
plt.rcParams['axes.unicode_minus'] = False

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'
# InteractiveShell.ast_node_interactivity = 'last'

import cv2 as cv

# import torch
# import torchvision
# import torch.nn as nn
# import torch.nn.functional as F
# import torch.utils.data as Data
# from torch.utils.data import DataLoader
# from torch.utils.data.dataset import Dataset

import pylatex
import latexify

import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```python
# TODO 日志、计时
# from colorama.Fore import RED, RESET
from colorama import Fore
```

```python
import logging
fmt = '%(asctime)s - %(levelname)8s - %(message)s'
formatter = logging.Formatter(fmt)
handler_control = logging.StreamHandler()  # stdout to console
handler_control.setLevel('INFO')  # 设置 INFO 级别
handler_control.setFormatter(formatter)

logger = logging.getLogger()
# logger.setLevel('INFO')
logger.addHandler(handler_control)

def timeit(text):
    def func_deco(func):
        """ 用来计时的装饰器函数 """
        def func_wrapper(*args, **kwargs):
            from time import time
            t0 = time()
#             logging.info(text + "开始计时")
            print(Fore.RED, text + "开始计时: ", Fore.RESET)
            res = func(*args, **kwargs)
            t1 = time()
#             logging.info(text + "用时: " + str(t1 - t0) + "s")
            print(Fore.RED, text + "结束计时，用时: ", str(t1 - t0), "s", Fore.RESET)
            return res
        return func_wrapper
    return func_deco
```

In [3]:

```python
# TODO DIR
ROOTDIR = pathlib.Path(os.path.abspath('.'))
IMG_HTML = ROOTDIR / 'img-html'
IMG_SVG = ROOTDIR / 'img-svg'
DATA_RAW = ROOTDIR / 'data-raw'
DATA_COOKED = ROOTDIR / 'data-processed'
```

In [4]:

```python
# TODO 附件4参数
浮子质量 = 4866  # kg
浮子底半径 = 1  # m
浮子圆柱部分高度 = 3  # m
浮子圆锥部分高度 = 0.8  # m
振子质量 = 2433  # kg
振子半径 = 0.5  # m
振子高度 = 0.5  # m
海水的密度 = 1025  # kg/m^3
重力加速度 = 9.8  # m/s^2
弹簧刚度 = 80000  # N/m
弹簧原长 = 0.5  # m
```

```
扭转弹簧刚度 = 250000  # N·m
静水恢复力矩系数 = 8890.7  # N·m
```

In [74]:

```
# TODO 附件3参数

class question1234:
    """设置具体问题几的参数"""
    def __init__(self, question):
        global 入射波浪频率
        global 垂荡附加质量
        global 纵摇附加转动惯量
        global 垂荡兴波阻尼系数
        global 纵摇兴波阻尼系数
        global 垂荡激励力振幅
        global 纵摇激励力矩振幅
        global 波浪频率
        global 波浪周期

        if question == 1:
            # 问题1: 参数
            # 纵摇附加转动惯量 = 6779.315  # kg·m^2
            # 纵摇兴波阻尼系数 = 151.4388  # N·m·s
            # 纵摇激励力矩振幅 = 1230  # N·m
            纵摇附加转动惯量 = None  # 问题1未使用，为避免使用/误用，初始化为 None
            纵摇兴波阻尼系数 = None  # 问题1未使用，为避免使用/误用，初始化为 None
            纵摇激励力矩振幅 = None  # 问题1未使用，为避免使用/误用，初始化为 None
            入射波浪频率 = 1.4005  # s^{-1}
            垂荡附加质量 = 1335.535  # kg
            垂荡兴波阻尼系数 = 656.3616  # N·s/m
            垂荡激励力振幅 = 6250  # N
            波浪频率 = 入射波浪频率
            波浪周期 = 1 / 波浪频率
        elif question == 2:
            # 问题2: 参数
            # 纵摇附加转动惯量 = 7131.29
            # 纵摇兴波阻尼系数 = 2992.724
            # 纵摇激励力矩振幅 = 2560
            纵摇附加转动惯量 = None  # 问题2未使用，为避免使用/误用，初始化为 None
            纵摇兴波阻尼系数 = None  # 问题2未使用，为避免使用/误用，初始化为 None
            纵摇激励力矩振幅 = None  # 问题2未使用，为避免使用/误用，初始化为 None
            入射波浪频率 = 2.2143
            垂荡附加质量 = 1165.992
            垂荡兴波阻尼系数 = 167.8395
            垂荡激励力振幅 = 4890
            波浪频率 = 入射波浪频率
            波浪周期 = 1 / 波浪频率
        elif question == 3:
            # 问题3：参数
            入射波浪频率 = 1.7152
            垂荡附加质量 = 1028.876
            纵摇附加转动惯量 = 7001.914
            垂荡兴波阻尼系数 = 683.4558
            纵摇兴波阻尼系数 = 654.3383
            垂荡激励力振幅 = 3640
            纵摇激励力矩振幅 = 1690
            波浪频率 = 入射波浪频率
            波浪周期 = 1 / 波浪频率
        elif question == 4:
            # 问题4：参数
            入射波浪频率 = 1.9806
            垂荡附加质量 = 1091.099
            纵摇附加转动惯量 = 7142.493
            垂荡兴波阻尼系数 = 528.5018
            纵摇兴波阻尼系数 = 1655.909
            垂荡激励力振幅 = 1760
            纵摇激励力矩振幅 = 2140
            波浪频率 = 入射波浪频率
            波浪周期 = 1 / 波浪频率

        return None

class trange:
    """设置时间区间和间隔的参数"""
    def __init__(self, left, right, step):
        global t_left
        global t_right
        global t_step

        t_left = left
        t_right = right
        t_step = step

        return None

_ = question1234(4)
_ = trange(0, 200, 0.2)
```

In [6]:

```
# TODO 跟变量有关的参数函数（这个后面有用，但是用处不大）

def S浮子底面积_func(r=浮子底半径):
    return np.pi * r**2
S浮子底面积 = S浮子底面积_func()
```

```python
def V排_func(h, pprint=True):
    """
    :param h: 圆柱壳体的入水深度
    :param pprint: 是否打印状态
    :return: V排 (m^3)
    """
    if h >= 0:
        print("圆锥壳体完全浸没")
        V排 = (1/3 * S浮子底面积 * 浮子圆锥部分高度) + (S浮子底面积 * h)
    else:
        print("圆锥壳体漂浮")
        depth = 浮子圆锥部分高度 + h
        r = 浮子底半径 * depth / 浮子圆锥部分高度
        V排 = 1/3 * S浮子底面积_func(r) * depth
    return V排
# print("浮子入水体积: ", V排_func(3))
# print("浮子入水体积: ", V排_func(2.4147))
# print("浮子入水体积: ", V排_func(0))
# print("浮子入水体积: ", V排_func(-0.001))
# print("浮子入水体积: ", V排_func(-0.8))

def F静水恢复力_func(h, pprint=False):
    """ 类似(就是)浮力 方向向上
    :param h: 圆柱壳体的入水深度
    :param pprint: 是否打印状态
    :return: F静水恢复力 (N)
    """
    F静水恢复力 = 海水的密度 * 重力加速度 * V排_func(h, pprint)
    return F静水恢复力
# F静水恢复力_func(2.4147)

def F兴波阻尼力_func(v, k=垂荡兴波阻尼系数):
    """ 方向同速度方向
    :param v: 速度
    :return:
    """
    F兴波阻尼力 = k * v
    return F兴波阻尼力

def F波浪激励力_func(t, omega=入射波浪频率, f=垂荡激励力振幅):
    """ 方向向上
    :param t: 时间
    :return: F波浪激励力 (N)
    """
    F波浪激励力 = f * np.cos(omega * t)
    return F波浪激励力
# F波浪激励力_func(0)

def F附加惯性力_func(m=垂荡附加质量, g=重力加速度):
    """ 方向向下 """
    F附加惯性力 = m * g
    return F附加惯性力
F附加惯性力 = F附加惯性力_func()

def F重力_func(m=浮子质量+振子质量, g=重力加速度):
    """ 方向向下 """
    F重力 = m * g
    return F重力
F重力 = F重力_func()

def c直线阻尼器的阻尼系数_func1():
    c直线阻尼器的阻尼系数 = 10000  # N·s/m
    return c直线阻尼器的阻尼系数

def c直线阻尼器的阻尼系数_func2(v浮子, v振子, k=10000, a=0.5):
    c直线阻尼器的阻尼系数 = k * abs(v浮子 - v振子)**a  # N·s/m
    return c直线阻尼器的阻尼系数
```

In [82]:

```
# TODO 问题3: 参数
m1, m2, me = 浮子质量, 振子质量, 垂荡附加质量
j2_func = lambda y3: ((0.969588 + y3)**3 - (0.469558 + y3)**3) * m2 / 1.5  # y3
j1, je = 33000, 纵摇附加转动惯量
omega = 入射波浪频率

c = 10000  # 直线阻尼器
f = 垂荡激励力振幅
k = 弹簧刚度
k1 = -垂荡兴波阻尼系数
k2 = -海水的密度 * 重力加速度 * S浮子底面积

C = 1000  # 旋转阻尼器
L = 纵摇激励力矩振幅
K = 扭转弹簧刚度
K1 = -纵摇兴波阻尼系数
K2 = -静水恢复力矩系数
```

查看何时稳定

大约在 $100s$ 左右稳定，取 $100-200s$

In [83]:

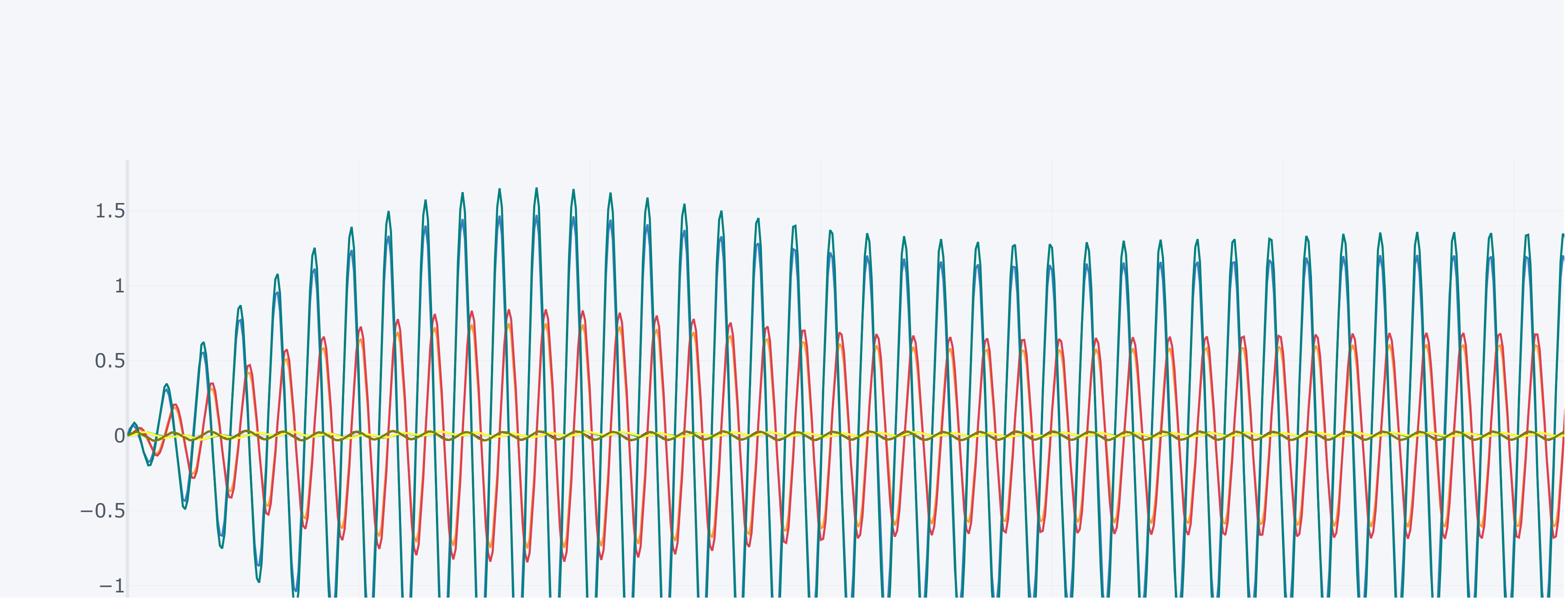
```
# TODO ode_func
c = 10000  # 直线阻尼器
C = 1000  # 旋转阻尼器

def ode_func(y, t):
    dy1 = y[2-1]
    dy3 = y[4-1]
    dy4 = c * (y[2-1] - y[4-1] * np.cos(y[7-1])) + k * (y[1-1] - y[3-1] * np.cos(y[7-1]))
    dy2 = f * np.cos(omega * t) + k1 * y[2-1] + k2 * y[1-1] - dy4
    dy2 /= m1 + me
    dy4 /= m2 * np.cos(y[7-1])

    j2 = j2_func(y[3-1])

    dy5 = y[6-1]
    dy7 = y[8-1]
    dy8 = C * (y[6-1] - y[8-1]) + K * (y[5-1] - y[7-1])
    dy6 = L * np.cos(omega * t) + K1 * y[6-1] + K2 * y[5-1] - dy8
    dy6 /= j1 + je
    dy8 /= j2
    return [dy1, dy2, dy3, dy4, dy5, dy6, dy7, dy8]
```

In [84]:

```
# TODO solve
t = np.linspace(t_left, t_right, num=int(t_right / t_step) + 1)
y0 = [0 for _ in range(8)]
res4 = odeint(ode_func, y0, t)
```

In [85]:

```
# TODO 绘图（不保存该结果）
def get_result4_df(t=t, result4=res4):
    columns = ['时间 (s)', '浮子垂荡位移 (m)', '浮子垂荡速度 (m/s)', '振子垂荡位移 (m)', '振子垂荡速度 (m/s)',
               '浮子纵摇角位移 (rad)', '浮子纵摇角速度 (rad/s)', '振子纵摇角位移 (rad)', '振子纵摇角速度 (rad/s)']
    shijian = pd.DataFrame(t, columns=columns[:1])
    result4 = pd.DataFrame(result4, columns=columns[1:])
    result4_df = pd.concat([shijian, result4], axis=1)
    result4_df = result4_df.iloc[:, [0, 1, 2, 5, 6, 3, 4, 7, 8]]
    return result4_df
result4_df = get_result4_df()
result4_df.iplot(x='时间 (s)', )
```



平均输出功率公式

平均输出功率：$P = \frac{1}{t_2 - t_1}\int_{t1}^{t2} F_G \Delta v + M_G \Delta \omega dt = \frac{1}{t_2 - t_1}\int_{t1}^{t2} c\Delta v^2 + C\Delta\omega^2 dt$

In [96]:

```
# TODO 数值解
def ode_func(y, t, c, C, k=k, K=K, k1=k1, k2=k2, K1=K1, K2=K2):
    dy1 = y[2-1]
    dy3 = y[4-1]
    dy4 = c * (y[2-1] - y[4-1] * np.cos(y[7-1])) + k * (y[1-1] - y[3-1] * np.cos(y[7-1]))
    dy2 = f * np.cos(omega * t) + k1 * y[2-1] + k2 * y[1-1] - dy4
    dy2 /= m1 + me
    dy4 /= m2 * np.cos(y[7-1])

    j2 = j2_func(y[3-1])

    dy5 = y[6-1]
    dy7 = y[8-1]
    dy8 = C * (y[6-1] - y[8-1]) + K * (y[5-1] - y[7-1])
    dy6 = L * np.cos(omega * t) + K1 * y[6-1] + K2 * y[5-1] - dy8
    dy6 /= j1 + je
    dy8 /= j2
    return [dy1, dy2, dy3, dy4, dy5, dy6, dy7, dy8]
def Power_func(res, c, C):
    """
    :param res: [X1, X1', X2, X2', theta1, theta1', theta2, theta2']
                [X1, V1, X2, V2, theta1, omega1, theta2, omega2]
    :param c: 直线
    :param C: 旋转
    :return: P
    """
    t_stable = 100
    begin = int(t_stable / t_step)
    end = int(t_right / t_step)
#     end = -2

    delta_V = (res[:, 1] - res[:, 3])**2
    delta_omega = (res[:, 5] - res[:, 7])**2
    power_i = c * delta_V + C * delta_omega
    power_ = power_i[begin:end+1]

#     power = power_ * t_step  # 矩形
#     P = power.sum() / (t_right - t_stable)
    power = (power_[1:] + power_[:-1]) * t_step / 2  # 梯形
    P = power.sum() / (t_right - t_stable - t_step)
#     print(P)

    return P
def LMS_func(c, C, pprint=False):
    res = odeint(
        ode_func,
        [0 for _ in range(8)],
        np.linspace(t_left, t_right, num=int(t_right / t_step) + 1),
        args=(c, C),
    )
    P = Power_func(res, c, C)
    if pprint:
        print("平均输出功率: ", P)
    return P
def GA_func(c, C, pprint=False):
    """ sko 遗传目标函数 """
    P = LMS_func(c, C)
    target = -P
    return target
def Program_func(x):
    return GA_func(x[0], x[1], pprint=False)
```

解微分方程组

In [109…

```
# TODO 求解器类
class Solver:
    def __init__(self):
        self.lb = 0
        self.ub = 1e5
        self.cs = None
        self.CS = None
        self.pss = None

        self.jit = False
        self.epoch = 100
        self.pprint = False

    # TODO 遗传
    @timeit("遗传")
    def run_GA(self):
        ga = GA(
            func=GA_func,
            n_dim=2,
            size_pop=50,
            max_iter=200,
            prob_mut=0.001,
            lb=[self.lb, self.lb],
            ub=[self.ub, self.ub],
            constraint_eq=tuple(),
            constraint_ueq=tuple(),
            precision=1e-06,
            early_stop=True,
        )
        xbest, ybest = ga.run()
        print("[直线阻尼器的阻尼系数, 旋转阻尼器的阻尼系数]: ", xbest, "最大输出功率", -ybest)
        return xbest, -ybest
    # TODO 规划
    @timeit("规划")
    def run_Program(self, x0=[50000, 50000]):
        """ 规划 Program """
        opt = minimize(
            Program_func,
            x0=x0,
            bounds=((self.lb, self.ub), (self.lb, self.ub), ),
            method='trust-constr',  #  SLSQP  trust-constr
        #     options={'xtol': 1e-30,  'gtol': 1e-30, 'disp': True},
        )
        xbest = opt.x
        ybest = -Program_func(xbest)
        print("[直线阻尼器的阻尼系数, 旋转阻尼器的阻尼系数]: ", xbest, "最大输出功率", ybest)
        return xbest, ybest

    # TODO 变步长
    @timeit("变步长")
    def run_LMS(self, *args, **kwargs):
        if self.jit:
            return self._run_LMS_jit(*args, **kwargs)
        else:
            return self._run_LMS_nojit(*args, **kwargs)

    def _find_LMS_best(self, cs, CS, pss):
        self.cs = cs
        self.CS = CS
        self.pss = pss
        index = np.unravel_index(self.pss.argmax(), self.pss.shape)
        return [self.cs[index[0]], self.CS[index[1]]], self.pss[index]

    def _LMS_main(self,
                  cs_lb=0, cs_ub=1e5, cs_num=100, cs_alpha=0,
                  CS_lb=0, CS_ub=1e5, CS_num=100, CS_alpha=0):
        t0 = time()
        batch_size = cs_num // self.epoch
        num_len = len(str(cs_num))
        fm = "%" + str(num_len) + "d/%" + str(num_len) + "d"

        if self.epoch is not None:  # 用户的分割
            batch_size = cs_num // self.epoch
            if batch_size == 0:  # 用户非法分割
                self.epoch = None
        if self.epoch is None:  # 自动处理非法分割
            batch_size = cs_num // 100  # 分割 100 份
            self.epoch = 100
            batch_size = cs_num // 10 if batch_size == 0 else batch_size  # 分割 10 份
            self.epoch = 10
            batch_size = cs_num // 1 if batch_size == 0 else batch_size  # 分割 1 份
            self.epoch = 1

        pss = np.zeros([cs_num+1, CS_num+1])

#         cs0 = 58467.62477501568
#         CS0 = 47662.22129472001

        cs_lb = cs_lb * (1 - cs_alpha)
        cs_ub = cs_ub * (1 + cs_alpha)
        CS_lb = CS_lb * (1 - CS_alpha)
        CS_ub = CS_ub * (1 + CS_alpha)

        cs = np.linspace(cs_lb, cs_ub, cs_num+1)
        CS = np.linspace(CS_lb, CS_ub, CS_num+1)
        for i, c in enumerate(cs):
            for j, C in enumerate(CS):
                pss[i, j] = LMS_func(c, C, pprint=False)

            if i % batch_size == 0:
                t1 = time()
                _stars = '[' + '*' * (i // batch_size) + '.' * ((cs_num - i) // batch_size) + ']'
                print(fm % (i, cs_num), _stars, round(t1 - t0, 2), "s/iter")
                t0 = time()
        xbest, ybest = self._find_LMS_best(cs, CS, pss)
        print("[直线阻尼器的阻尼系数, 旋转阻尼器的阻尼系数]: ", xbest, "最大输出功率", ybest)
        return xbest, ybest
    def _run_LMS_nojit(self, *args, **kwargs):
        return self._LMS_main(*args, **kwargs)
    @jit
    def _run_LMS_jit(self, *args, **kwargs):
        return self._LMS_main(*args, **kwargs)

    # TODO 绘图
    def _plot_Surface(self, show=True, save=False):
        fig = go.Figure(data=[go.Surface(
            x=self.cs,
            y=self.CS,
            z=self.pss,
        )])

        if save:
            fig.write_image(IMG_SVG / '问题4-3D曲线图.svg')
        if show:
            fig.show()
        return True
    def _plt_contourf(self, show=True, save=False):
        mi, mx = self.pss.min(), self.pss.max()
        lb1, ub1, lb2, ub2 = self.cs.min(), self.cs.max(), self.CS.min(), self.CS.max()

        plt.figure(dpi=100)
        plt.contourf(self.pss.T, np.arange(mi, mx+1, (mx - mi) / 100), origin='lower', extent=[lb1, ub1, lb2, ub2], cmap=plt.cm.jet)
        plt.xlabel('直线阻尼器阻尼系数')
        plt.ylabel('旋转阻尼器阻尼系数')
        plt.title("平均输出功率")

        if save:
            plt.savefig(IMG_SVG / '问题4-二维等高线图.svg')
        if show:
            plt.show()
        return True
    def plot_result(self, *args, **kwargs):
        self._plot_Surface(*args, **kwargs)
        self._plt_contourf(*args, **kwargs)
        return None
```

In [118…

```
# TODO 单独运行求解
_ = question1234(4)
_ = trange(0, 150, 0.2)
solver = Solver()

solver.run_GA()  # 遗传
solver.run_Program()  # 规划
solver.run_LMS(cs_num=10, CS_num=10)  # 变步长
# solver.run_LMS(cs_num=100, CS_num=100)  # 遍历：一般考验电脑性能，预计用时：∞（没试过不知道）
# solver.run_LMS(cs_num=1000, CS_num=1000)  # 遍历：比较考验电脑性能，预计用时：∞（没试过不知道）
# solver.run_LMS(cs_num=10000, CS_num=10000)  # 遍历：究极考验电脑性能，预计用时：∞（没试过不知道）
# solver.run_LMS(cs_num=100000, CS_num=100000)  # 遍历：地狱考验电脑性能，预计用时：∞（没试过不知道）

solver.plot_result(show=True, save=False)  # 画图
```

```
 遗传开始计时：
[直线阻尼器的阻尼系数，旋转阻尼器的阻尼系数]:  [57536.87948714 31004.29883511] 最大输出功率 [317.69786004]
 遗传结束计时，用时： 25.64826989173889 s
```

Out[118]:

```
(array([57536.87948714, 31004.29883511]), array([317.69786004]))
```

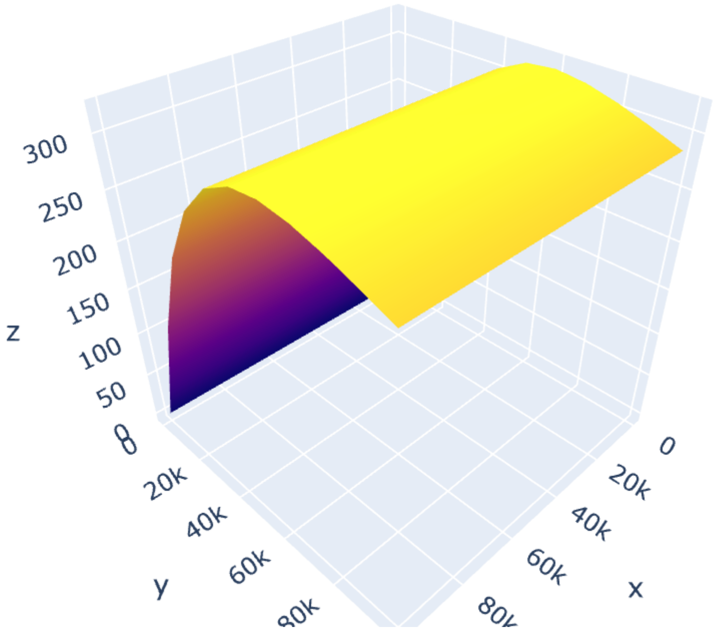
```
 规划开始计时：
[直线阻尼器的阻尼系数，旋转阻尼器的阻尼系数]:  [50000.23073816 49999.8185654 ] 最大输出功率 314.74575018315727
 规划结束计时，用时： 111.9183702468872 s
```

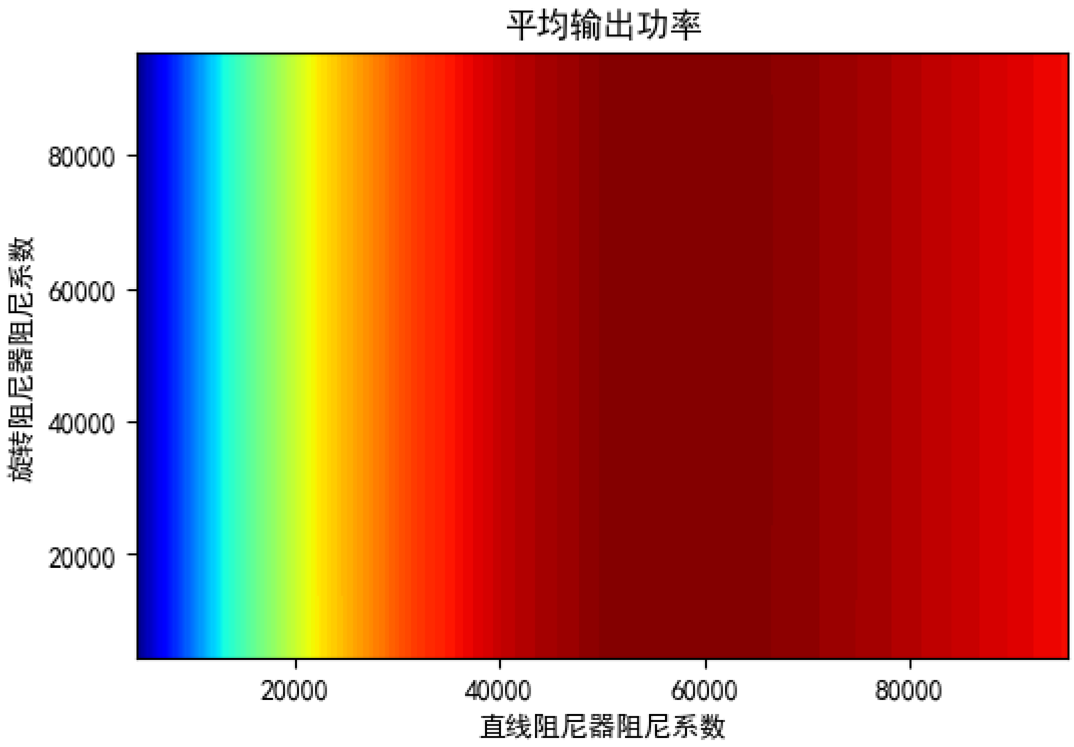
Out[118]:

```
(array([50000.23073816, 49999.8185654 ]), 314.74575018315727)
```

```
 变步长开始计时：
 0/10 [..........] 7.16 s/iter
 1/10 [*.........] 6.25 s/iter
 2/10 [**........] 6.17 s/iter
 3/10 [***.......] 6.41 s/iter
 4/10 [****......] 6.48 s/iter
 5/10 [*****.....] 6.14 s/iter
 6/10 [******....] 6.22 s/iter
 7/10 [*******...] 6.35 s/iter
 8/10 [********..] 7.07 s/iter
 9/10 [*********.] 7.46 s/iter
10/10 [**********] 7.19 s/iter
[直线阻尼器的阻尼系数，旋转阻尼器的阻尼系数]：  [60000.0, 50000.0] 最大输出功率 317.6470046262132
 变步长结束计时，用时： 72.915207862854 s
```

```
Out[118]: ([60000.0, 50000.0], 317.6470046262132)
```


z
300
250
200
150
100
50
0
0
20k
40k
60k
80k
y
0
20k
40k
60k
80k
x



In [117…

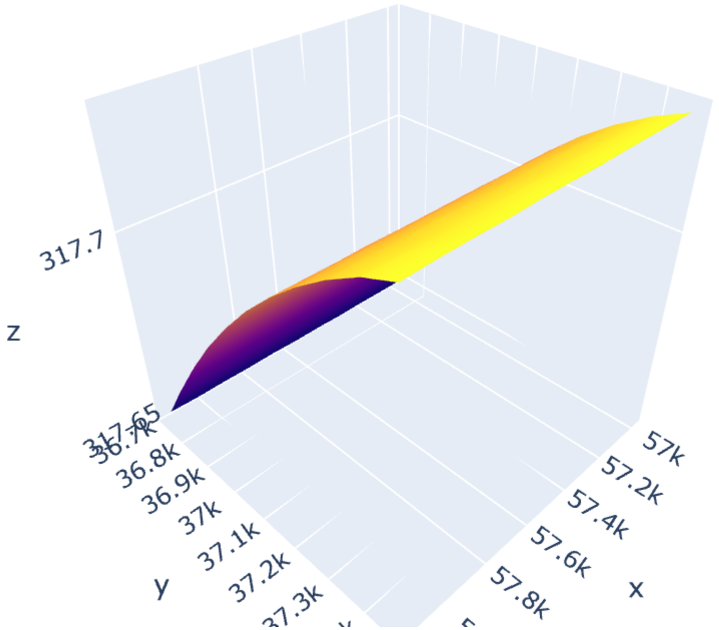
```
# TODO 联合运行求解
_ = question1234(4)
_ = trange(0, 150, 0.2)
solver = Solver()
solver.epoch = 10
solver.pprint = False
solver.jit = False

# TODO GA + LMS
t0 = time()
xbest, ybest = solver.run_GA()  # 遗传
# xbest = (58467.64816206559, 47662.22129472)
solver.run_LMS(cs_lb=xbest[0], cs_ub=xbest[0], cs_num=10, cs_alpha=0.1,
               CS_lb=xbest[1], CS_ub=xbest[1], CS_num=10, CS_alpha=0.1)  # 变步长
print("总用时: ", time() - t0)
solver.plot_result(show=True, save=False)  # 绘图

print('-' * 100)

# TODO Program + LMS
t0 = time()
xbest, ybest = solver.run_Program()  # 规划
```

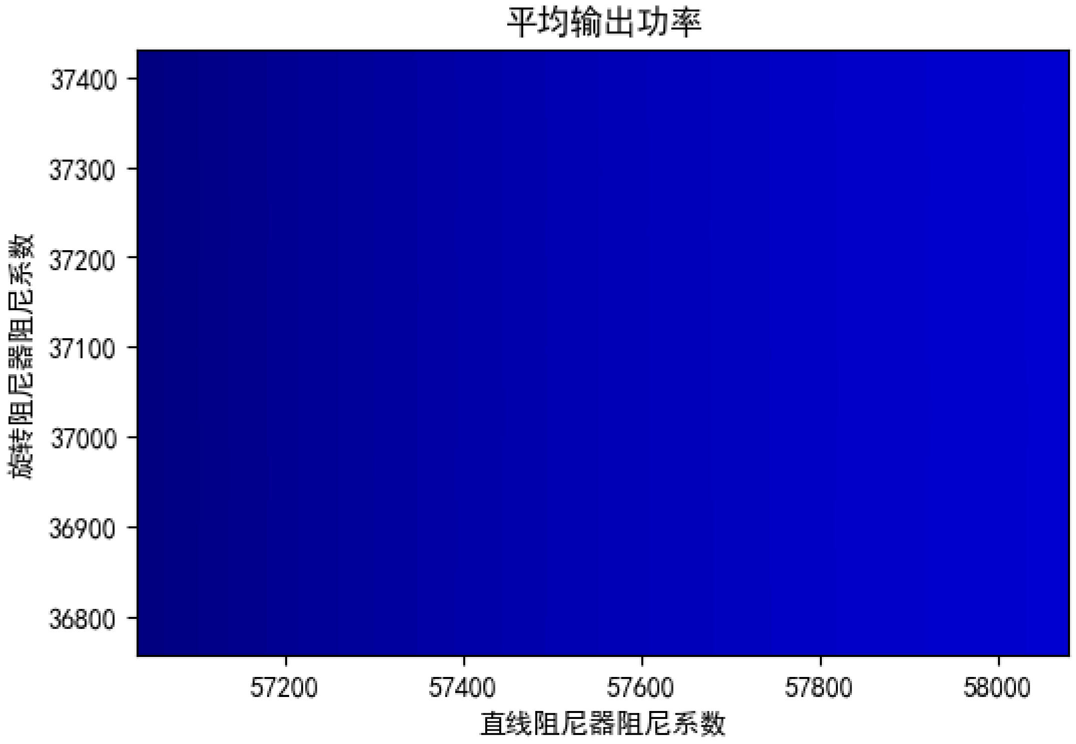
```
# xbest = (58467.64816206559, 47662.22129472)
solver.run_LMS(cs_lb=xbest[0], cs_ub=xbest[0], cs_num=10, cs_alpha=0.1,
               CS_lb=xbest[1], CS_ub=xbest[1], CS_num=10, CS_alpha=0.1)  # 变步长
print("总用时: ", time() - t0)
solver.plot_result(show=True, save=False)  # 绘图
```

```
 遗传开始计时:
[直线阻尼器的阻尼系数, 旋转阻尼器的阻尼系数]:  [57556.41995171 37094.13072674] 最大输出功率 [317.70030272]
 遗传结束计时, 用时:  26.062267065048218 s
 变步长开始计时:
 0/10 [..........] 5.01 s/iter
 1/10 [*.........] 5.0 s/iter
 2/10 [**........] 5.01 s/iter
 3/10 [***.......] 4.94 s/iter
 4/10 [****......] 5.07 s/iter
 5/10 [*****.....] 5.12 s/iter
 6/10 [******....] 5.23 s/iter
 7/10 [*******...] 5.16 s/iter
 8/10 [********..] 5.27 s/iter
 9/10 [*********.] 5.13 s/iter
10/10 [**********] 4.93 s/iter
[直线阻尼器的阻尼系数, 旋转阻尼器的阻尼系数]:  [58131.98415122411, 37316.695511103295] 最大输出功率 317.7262368268232
 变步长结束计时, 用时:  55.871594190597534 s
```

```
Out[117]: ([58131.98415122411, 37316.695511103295], 317.7262368268232)
```

```
总用时:  81.93683743476868
```

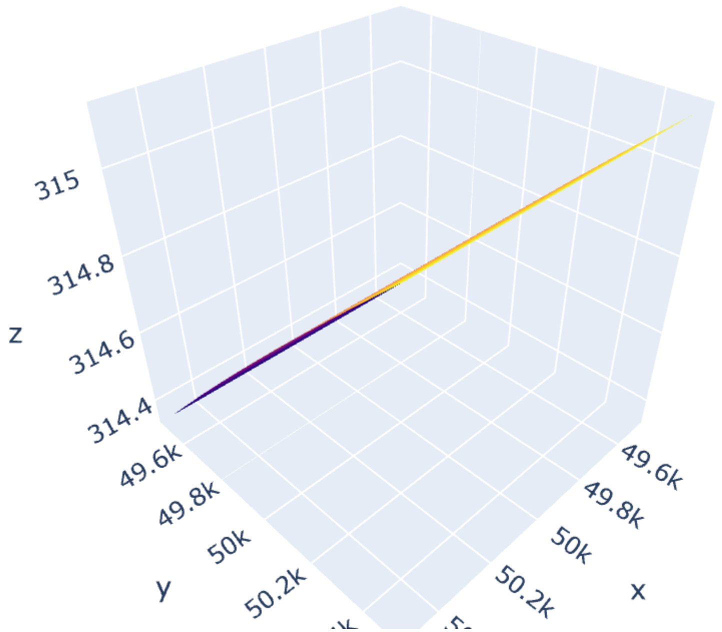

317.7
z
317.65
36.7k
36.8k
36.9k
37k
37.1k
37.2k
37.3k
y
57k
57.2k
57.4k
57.6k
57.8k
x



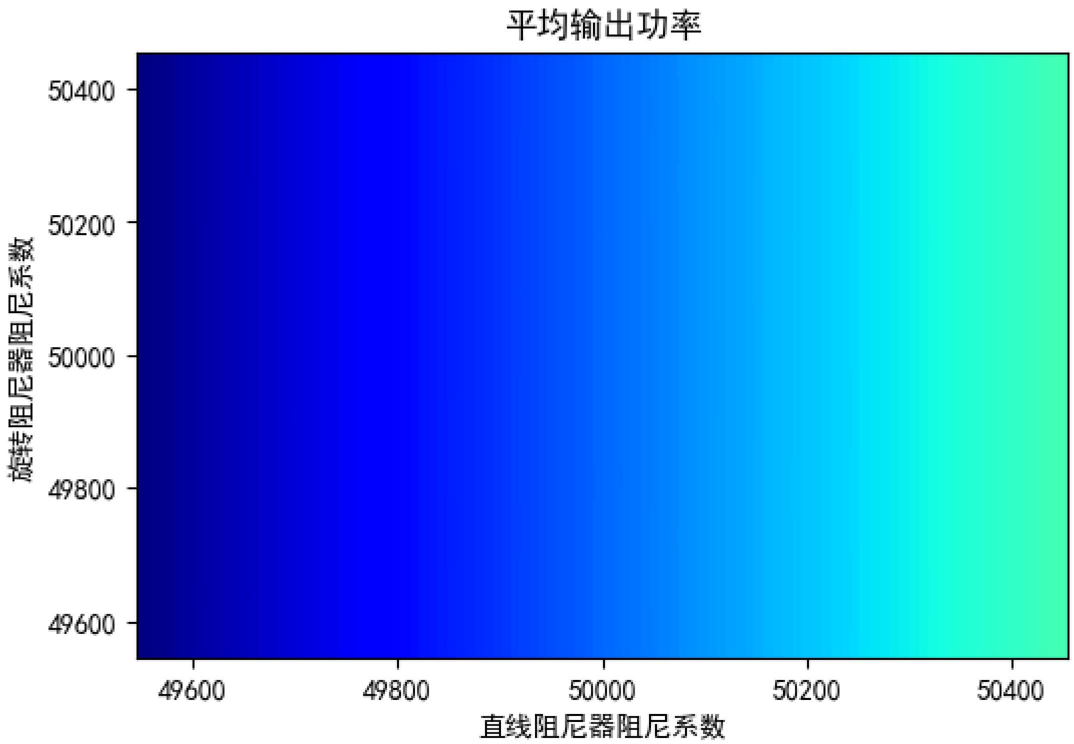
```
-----------------------------------------------------------------------------------------------------
 规划开始计时：
[直线阻尼器的阻尼系数，旋转阻尼器的阻尼系数]：  [50000.23073816 49999.8185654 ] 最大输出功率 314.74575018315727
 规划结束计时，用时：  126.50902438163757 s
 变步长开始计时：
 0/10 [..........] 6.46 s/iter
 1/10 [*.........] 5.82 s/iter
 2/10 [**........] 5.26 s/iter
 3/10 [***.......] 5.65 s/iter
 4/10 [****......] 5.61 s/iter
 5/10 [*****.....] 5.31 s/iter
 6/10 [******....] 5.57 s/iter
 7/10 [*******...] 7.26 s/iter
 8/10 [********..] 5.92 s/iter
 9/10 [*********.] 5.25 s/iter
10/10 [**********] 5.21 s/iter
[直线阻尼器的阻尼系数，旋转阻尼器的阻尼系数]：  [50500.23304554521, 49499.82037974576] 最大输出功率 315.11129621858487
 变步长结束计时，用时：  63.31871557235718 s
```

```
Out[117]: ([50500.23304554521, 49499.82037974576], 315.11129621858487)
```

```
总用时：  189.8307318687439
```



315
314.8
314.6
314.4
z
49.6k
49.8k
50k
50.2k
y
49.6k
49.8k
50k
50.2k
x


平均输出功率
50400
50200
50000
49800
49600
旋转阻尼器阻尼系数
49600
49800
50000
50200
50400
直线阻尼器阻尼系数

In [ ]:

In [ ]:

In [ ]: