问题2

分别计算两种情况的最大输出功率及相应的最优阻尼系数

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

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]:

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

```
# 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]:

```
# 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 [5]:

```
# 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(2)
_ = 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()

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
```

平均输出功率公式

$$P = \frac{1}{t_2 - t_1}\int_{t1}^{t2} F_G \Delta v dt = \frac{1}{t_2 - t_1}\int_{t1}^{t2} c\Delta v^2 dt$$

准备

In [7]:

```
# TODO parameters and func
m1 = 浮子质量
m2 = 振子质量
m1_ = 浮子质量 + 垂荡附加质量
m2_ = 振子质量 + 垂荡附加质量
k = 弹簧刚度
c直线阻尼器的阻尼系数 = c直线阻尼器的阻尼系数_func1()
c = c直线阻尼器的阻尼系数
f = 垂荡激励力振幅
omega = 入射波浪频率
k1 = -垂荡兴波阻尼系数
```

```python
k2 = -海水的密度 * 重力加速度 * S浮子底面积

def differential_equations_1(ys, t, c=c, k=k, k1=k1, k2=k2):
    """ 第1小问的方程求解函数 """
    y1 = ys[2-1]
    y3 = ys[4-1]

    y4 = c * (ys[2-1] - ys[4-1]) + k * (ys[1-1] - ys[3-1])
    y2 = f * np.cos(omega * t) + k1 * ys[2-1] + k2 * ys[1-1] - y4

    y2 = y2 / m1_
    y4 = y4 / m2
    return [y1, y2, y3, y4]

def differential_equations_2(ys, t, xishu, mici, k=k, k1=k1, k2=k2):
    """ 第2小问的方程求解函数 """
    y1 = ys[2-1]
    y3 = ys[4-1]

    c = c直线阻尼器的阻尼系数_func2(ys[2-1], ys[4-1], k=xishu, a=mici)
    y4 = c * (ys[2-1] - ys[4-1]) + k * (ys[1-1] - ys[3-1])
    y2 = f * np.cos(omega * t) + k1 * ys[2-1] + k2 * ys[1-1] - y4

    y2 = y2 / m1_
    y4 = y4 / m2
    return np.array([y1, y2, y3, y4])

def get_power(task,
              pprint=False,
              t_left=0,
              t_right=100,
              t_step=None,
              stable_time_begin=60,
              stable_time_end=100,
              c=c, k=k, k1=k1, k2=k2,
              xishu=10000, mici=0.5,
              y0=[0 for _ in range(4)]):
    """ 获得平均输出功率（该函数与其他函数独立）
    :param task: 第几小问
    :param t_left: 设置为 0, 固定值
    :param t_right: 设置为 100, 固定值
    :param t_step: 设置为 0.2, 非固定值, 可以改
        时间间隔, 时间间隔越小结果越准确
    """
    t_left = 0
    t_right = 100
    t_step = 0.2 if t_step is None else t_step
    t = np.linspace(t_left, t_right, num=int(t_right / t_step) + 1)
    if task == 1:
        result1 = odeint(differential_equations_1, y0, t, args=(c, ))
    elif task == 2:
        result1 = odeint(differential_equations_2, y0, t, args=(xishu, mici, ))

    delta = abs(result1[:, 1] - result1[:, 3])  # 相对速度
    if task == 1:
        c = c
    elif task == 2:
        c = xishu * delta**mici
    F_zuli = c * delta
    power_i = F_zuli * delta

#     power_i = power_i[1:] * t_step  # 矩形
    power_i = (power_i[1:] + power_i[:-1]) * t_step / 2  # 梯形

    idx_begin = int(stable_time_begin / t_step)
    idx_end = int(stable_time_end / t_step)
    stable_time_length = stable_time_end - stable_time_begin

    power = power_i[idx_begin:idx_end]
    P = power.sum() / stable_time_length
    if pprint:
        print("平均输出功率：", P)
    return P
```

(1) c 常量

$c = 10000$

In [8]:

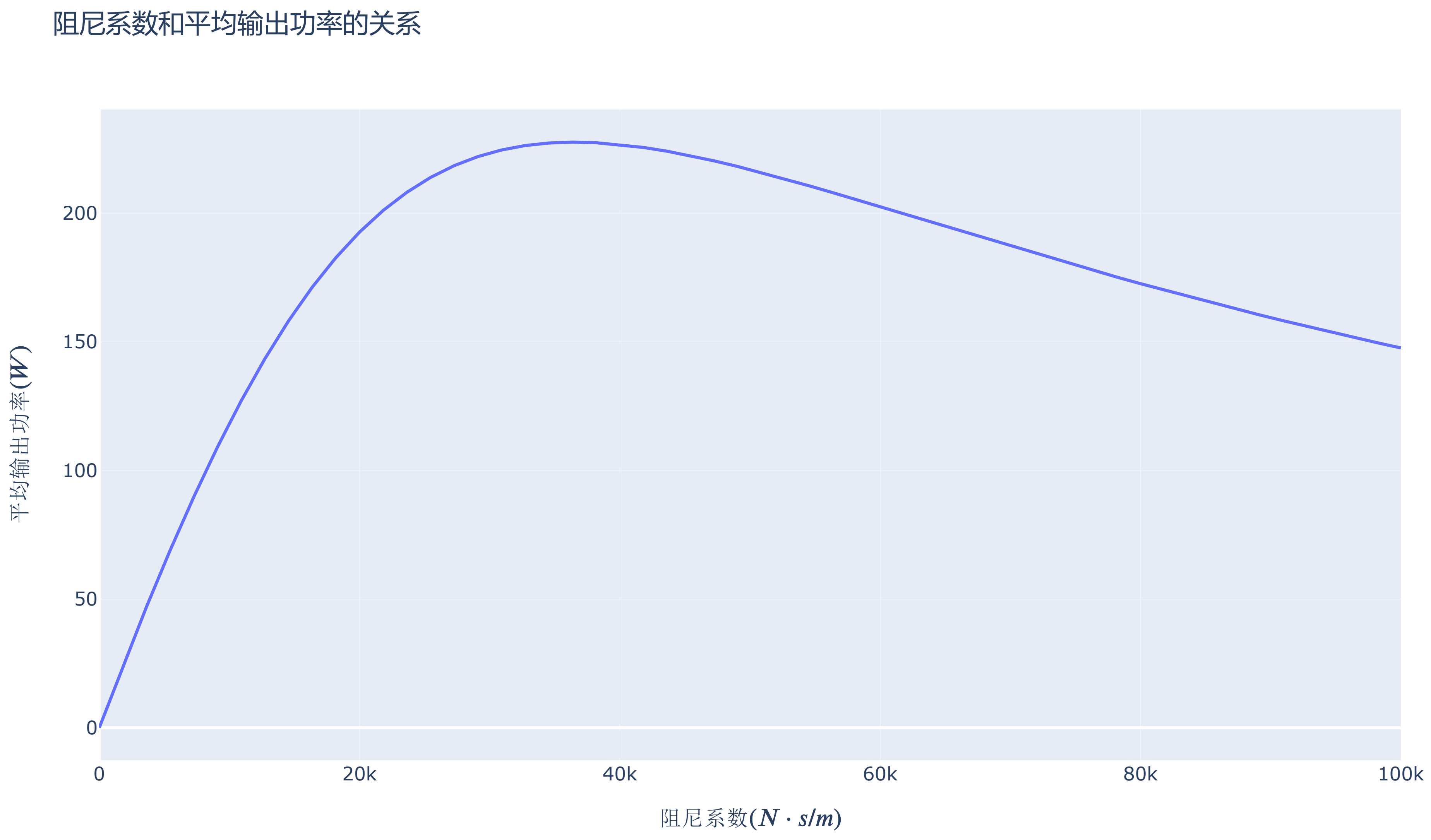
```
# TODO 第1小问求解类
class solver_task1:
    """ 第 1 小问求解 """
    def __init__(self):
        self.task = 1
        self.pprint = False
        self.epoch = None
        self.c_range = None
        self.ps = None
        self.jit = True

    # TODO 遗传
    def _GA_func(self, c):
        return -get_power(task=self.task, pprint=self.pprint, c=c)
    @timeit(text="遗传")
    def run_GA(self):
        """ 遗传 GA """
        lb, ub = 0, 1e5
        ga = GA(
            func=self._GA_func,
            n_dim=1,
            size_pop=50,
            max_iter=200,
            prob_mut=0.001,
            lb=[lb],
            ub=[ub],
            constraint_eq=tuple(),
            constraint_ueq=tuple(),
            precision=1e-07,
            early_stop=True,
        )
        xbest, ybest = ga.run()
        print("直线阻尼器的阻尼系数: ", xbest, "最大输出功率", -ybest)
        return float(xbest), -float(ybest)

    # TODO 规划
    def _Program_func(self, c):
        return -get_power(task=self.task, pprint=self.pprint, c=c)
    @timeit(text="规划")
    def run_Program(self, x0=35000):
        """ 规划 Program """
        lb, ub = 0, 1e5
        opt = minimize(
            self._Program_func,
            x0=x0,
            bounds=((lb, ub), ),
            method='trust-constr',  #  SLSQP  trust-constr
        #     options={'xtol': 1e-30,  'gtol': 1e-30, 'disp': True},
        )
        xbest = opt.x
        ybest = -self._Program_func(xbest)
        print("直线阻尼器的阻尼系数: ", xbest, "最大输出功率", ybest)
        return float(xbest), float(ybest)

    # TODO 变步长
    def _LMS_func(self, c):
        return get_power(task=self.task, pprint=self.pprint, c=c)
    def _find_LMS_best(self, c_range, ps):
        self.c_range = c_range
        self.ps = ps
        return c_range[np.argmax(ps)], np.max(ps)
    @timeit(text="变步长")
    def run_LMS(self, lb, ub, num, alpha=0):
        func = self._run_LMS_jit if self.jit else self._run_LMS_nojit
        return func(lb, ub, num, alpha)
    def _LMS_main(self, lb, ub, num, alpha):
        """ 变步长 LMS """
        num_len = len(str(num))
        fm = "%" + str(num_len) + "d/%" + str(num_len) + "d"
        if self.epoch is not None:  # 用户的分割
            batch_size = 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 = num // 100  # 分割 100 份
            self.epoch = 100
            batch_size = num // 10 if batch_size == 0 else batch_size  # 分割 10 份
            self.epoch = 10
            batch_size = num // 1 if batch_size == 0 else batch_size  # 分割 1 份
            self.epoch = 1

        # ps = [0 for _ in range(num + 1)]  # list
        ps = np.zeros(num + 1)  # np.ndarray
        t0 = time()
        lb = lb * (1 - alpha)
        ub = ub * (1 + alpha)
        print(f"变步长范围：[", lb, ',', ub, "]: ")
        c_range = np.linspace(lb, ub, num + 1)
        for i, c in enumerate(c_range):
            p = self._LMS_func(c)
            ps[i] = p  # list / np.ndarray

            # info
            if i % batch_size == 0:
                t1 = time()
                _stars = '[' + '*' * (i // batch_size) + '.' * ((num - i) // batch_size) + ']'
                print(fm % (i, num), _stars, round(t1 - t0, 2), "s/iter")
                t0 = time()

        xbest, ybest = self._find_LMS_best(c_range, ps)
        return xbest, ybest
    def _run_LMS_nojit(self, lb, ub, num, alpha):
        return self._LMS_main(lb, ub, num, alpha)
    @jit
    def _run_LMS_jit(self, lb, ub, num, alpha):
        return self._LMS_main(lb, ub, num, alpha)

    # TODO 绘图
    def plot_result(self, show=True, save=False):
        # TODO 阻尼系数和平均输出功率的关系
        x = self.c_range
        y = self.ps
        trace = go.Scatter(x=x, y=y)
        fig = go.Figure(data=trace)
        fig.update_layout(
            width=1000,
            height=600,
            xaxis=dict(title='$阻尼系数 (N·s/m)$'),
            yaxis=dict(title='$平均输出功率 (W)$'),
            title=dict(text='阻尼系数和平均输出功率的关系'),
        )
        if save:
            fig.write_image(IMG_SVG / "问题2-阻尼系数和平均输出功率的关系.svg")
        if show:
            fig.show()
        return None
```

In [9]:

```
# TODO 单独运行求解
solver1 = solver_task1()
solver1.pprint = False
solver1.epoch = 11
solver1.jit = True

# xbest, ybest = solver1.run_GA()  # 遗传
# solver1.run_Program(x0=35000)  # 规划
solver1.run_LMS(0, 100000, 55, alpha=0)  # 变步长

solver1.plot_result(show=True, save=True)  # 绘图
```

```
 变步长开始计时：
变步长范围：[ 0 , 100000 ]：
 0/55 [...........] 0.21 s/iter
 5/55 [*..........] 0.44 s/iter
10/55 [**.........] 0.33 s/iter
15/55 [***........] 0.29 s/iter
20/55 [****.......] 0.39 s/iter
25/55 [*****......] 0.56 s/iter
30/55 [******.....] 0.71 s/iter
35/55 [*******....] 0.72 s/iter
40/55 [********...] 0.67 s/iter
45/55 [*********..] 0.55 s/iter
50/55 [**********.] 0.52 s/iter
55/55 [***********] 0.58 s/iter
 变步长结束计时，用时： 6.5473363399505615 s
```

```
Out[9]: (36363.63636363637, 227.61665763794673)
```



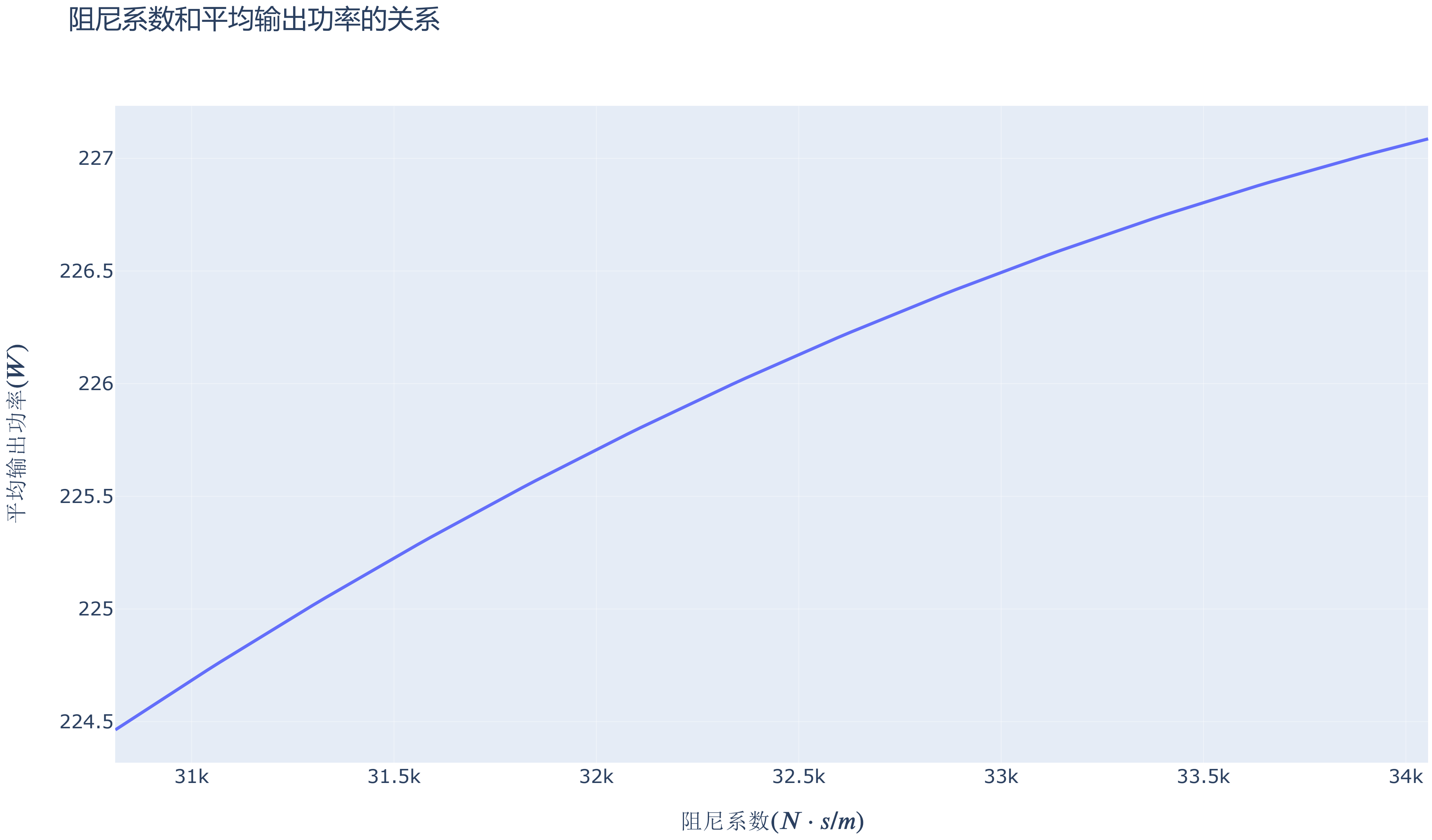
In [11]:

```
# TODO 联合运行求解
solver1 = solver_task1()
solver1.pprint = False
solver1.epoch = 24
solver1.jit = True

# TODO GA + LMS
t0 = time()
```

```
xbest, ybest = solver1.run_GA()  # 遗传
solver1.run_LMS(xbest, xbest, 100, alpha=0.05)  # 变步长
print("总用时: ", time() - t0)
solver1.plot_result(show=True, save=False)  # 绘图

print('-' * 100)

# TODO Program + LMS
t0 = time()
xbest, ybest = solver1.run_Program(x0=35000)  # 规划
solver1.run_LMS(xbest, xbest, 100, alpha=0.05)  # 变步长
print("总用时: ", time() - t0)
solver1.plot_result(show=True, save=False)  # 绘图
```

```
 遗传开始计时：
直线阻尼器的阻尼系数： [32433.27713756] 最大输出功率 [226.07675711]
 遗传结束计时，用时： 7.074131727218628 s
 变步长开始计时：
变步长范围：[ 30811.613280685207 , 34054.940994441546 ]：
  0/100 [.........................] 0.08 s/iter
  4/100 [*........................] 0.25 s/iter
  8/100 [**.......................] 0.22 s/iter
 12/100 [***......................] 0.25 s/iter
 16/100 [****.....................] 0.31 s/iter
 20/100 [*****....................] 0.2 s/iter
 24/100 [******...................] 0.28 s/iter
 28/100 [*******..................] 0.3 s/iter
 32/100 [********.................] 0.25 s/iter
 36/100 [*********................] 0.29 s/iter
 40/100 [**********...............] 0.25 s/iter
 44/100 [***********..............] 0.3 s/iter
 48/100 [************.............] 0.27 s/iter
 52/100 [*************............] 0.24 s/iter
 56/100 [**************...........] 0.24 s/iter
 60/100 [***************..........] 0.3 s/iter
 64/100 [****************.........] 0.29 s/iter
 68/100 [*****************........] 0.22 s/iter
 72/100 [******************.......] 0.23 s/iter
 76/100 [*******************......] 0.26 s/iter
 80/100 [********************.....] 0.29 s/iter
 84/100 [*********************....] 0.3 s/iter
 88/100 [**********************...] 0.38 s/iter
 92/100 [***********************..] 0.34 s/iter
 96/100 [************************.] 0.32 s/iter
100/100 [*************************] 0.38 s/iter
 变步长结束计时，用时： 7.122573614120483 s
```

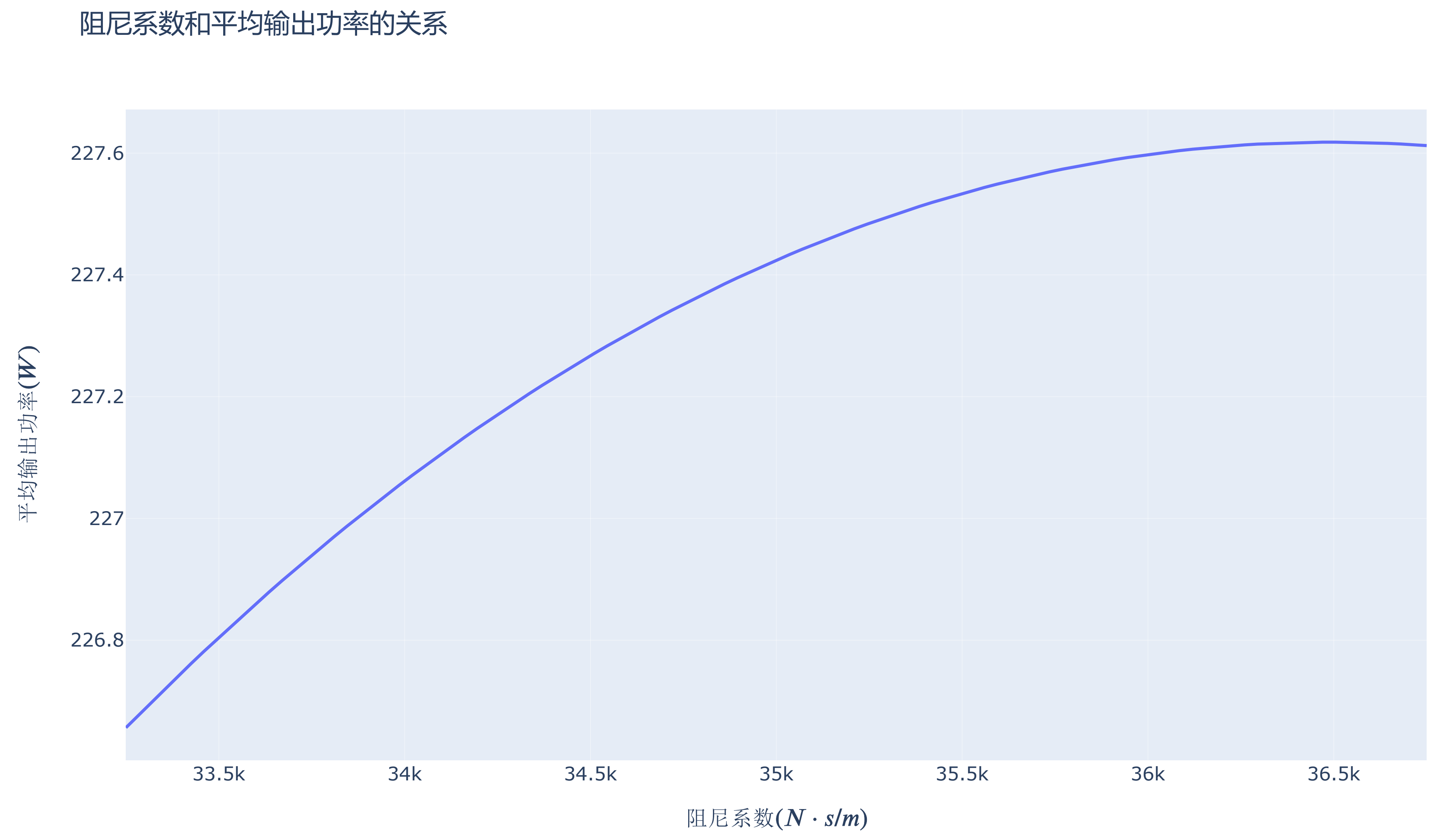
```
Out[11]: (34054.940994441546, 227.08648987690094)
         总用时：  14.200635194778442
```


阻尼系数和平均输出功率的关系

```
----------------------------------------------------------------------------------------------------
 规划开始计时：
直线阻尼器的阻尼系数：  [34999.975705] 最大输出功率 227.42405210598844
 规划结束计时，用时：  13.769381999969482 s
 变步长开始计时：
变步长范围：[ 33249.97691975324 , 36749.97449025358 ]：
  0/100 [.........................] 0.06 s/iter
  4/100 [*........................] 0.27 s/iter
  8/100 [**.......................] 0.35 s/iter
 12/100 [***......................] 0.38 s/iter
 16/100 [****.....................] 0.33 s/iter
 20/100 [*****....................] 0.35 s/iter
 24/100 [******...................] 0.32 s/iter
 28/100 [*******..................] 0.3 s/iter
 32/100 [********.................] 0.3 s/iter
 36/100 [*********................] 0.3 s/iter
 40/100 [**********...............] 0.4 s/iter
 44/100 [***********..............] 0.41 s/iter
 48/100 [************.............] 0.3 s/iter
 52/100 [*************............] 0.35 s/iter
 56/100 [**************...........] 0.29 s/iter
 60/100 [***************..........] 0.38 s/iter
 64/100 [****************.........] 0.38 s/iter
 68/100 [*****************........] 0.44 s/iter
 72/100 [******************.......] 0.36 s/iter
 76/100 [*******************......] 0.39 s/iter
 80/100 [********************.....] 0.43 s/iter
 84/100 [*********************....] 0.35 s/iter
 88/100 [**********************...] 0.41 s/iter
 92/100 [***********************..] 0.36 s/iter
 96/100 [************************.] 0.37 s/iter
100/100 [*************************] 0.35 s/iter
 变步长结束计时，用时：  8.927614450454712 s
```

```
Out[11]: (36504.974660318556, 227.6179770710277)
```

```
总用时：  22.699984073638916
```


阻尼系数和平均输出功率的关系

（2）c 非常量

$$c = 10000|V|^{0.5} = 10000|V_1 - V_2|^{0.5} = 10000|\dot{X}_1 - \dot{X}_2|^{0.5}$$

In [12]:

```python
# TODO 第2小问求解
class solver_task2:
    """ 第 2 小问求解 """
    def __init__(self):
        self.task = 2
        self.pprint = False
        self.epoch = 100
        self.xishu_s = None
        self.mici_s = None
        self.pss = None
        self.jit = True

    # TODO 遗传
    @staticmethod
    def _GA_func(xishu, mici):
        return -get_power(task=2, pprint=False, xishu=xishu, mici=mici)
    @timeit("遗传")
    def run_GA(self):
        """ 遗传 GA """
        lb, ub = 0, 1e5
        ga = GA(
            func=self._GA_func,
            n_dim=2,
            size_pop=50,
            max_iter=200,
            prob_mut=0.001,
            lb=[lb, 0],
            ub=[ub, 1],
            constraint_eq=tuple(),
            constraint_ueq=tuple(),
            precision=1e-07,
            early_stop=True,
        )
        xbest, ybest = ga.run()
        print("[比例系数, 幂指数]: ", xbest, "最大输出功率", -ybest)
        return xbest, -ybest


    # TODO 规划
    def _Program_func(self, x):
        return -get_power(task=self.task, pprint=self.pprint, xishu=x[0], mici=x[1])
    @timeit("规划")
    def run_Program(self, x0=[80000, 0.4]):
        """ 规划 Program """
        opt = minimize(
            self._Program_func,
            x0=x0,
            bounds=((0, 100000), (0, 1)),
            method='SLSQP',  #  SLSQP  trust-constr
            options={'xtol': 1e-5,  'gtol': 1e-5, 'disp': True}
        )
        xbest = opt.x
        ybest = -self._Program_func(xbest)
        print("[比例系数, 幂指数]: ", xbest, "最大输出功率", ybest)
        return xbest, ybest


    # TODO 变步长
    def _LMS_func(self, xishu, mici):
        return get_power(task=self.task, pprint=self.pprint, xishu=xishu, mici=mici)
    def _find_LMS_best(self, xishu_s, mici_s, pss):
        self.xishu_s = xishu_s  # np.ndarray
        self.mici_s = mici_s  # np.ndarray
        self.pss = pss  # np.ndarray
        index = np.unravel_index(self.pss.argmax(), self.pss.shape)
        return [self.xishu_s[index[0]], self.mici_s[index[1]]], self.pss[index]
    @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)
        func = self._run_LMS_jit if self.jit else self._run_LMS_nojit
        return func(*args, **kwargs)
    def _LMS_main(self,
                  xishu_lb=0, xishu_ub=100000, xishu_num=100, xishu_alpha=0,
                  mici_lb=0, mici_ub=1, mici_num=10, mici_alpha=0):
        """ 变步长 LMS """
        t0 = time()
        batch_size = xishu_num // self.epoch
        num_len = len(str(xishu_num))
        fm = "%" + str(num_len) + "d/%" + str(num_len) + "d"

        if self.epoch is not None:  # 用户的分割
            batch_size = xishu_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 = xishu_num // 100  # 分割 100 份
            self.epoch = 100
            batch_size = xishu_num // 10 if batch_size == 0 else batch_size  # 分割 10 份
            self.epoch = 10
            batch_size = xishu_num // 1 if batch_size == 0 else batch_size  # 分割 1 份
            self.epoch = 1
```

```python
        pss = np.zeros([xishu_num+1, mici_num+1])

        xishu_lb = xishu_lb * (1 - xishu_alpha)
        xishu_ub = xishu_ub * (1 + xishu_alpha)
        mici_lb = mici_lb * (1 - mici_alpha)
        mici_ub = mici_ub * (1 + mici_alpha)

        xishu_s = np.linspace(xishu_lb, xishu_ub, xishu_num+1)
        mici_s = np.linspace(mici_lb, mici_ub, mici_num+1)
        for i, xishu in enumerate(xishu_s):
            for j, mici in enumerate(mici_s):
                pss[i, j] = self._LMS_func(xishu=xishu, mici=mici)

            if i % batch_size == 0:
                t1 = time()
                _stars = '[' + '*' * (i // batch_size) + '.' * ((xishu_num - i) // batch_size) + ']'
                print(fm % (i, xishu_num), _stars, round(t1 - t0, 2), "s/iter")
                t0 = time()

        xbest, ybest = self._find_LMS_best(xishu_s, mici_s, pss)
        print("[比例系数, 幂指数]: ", 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 _plt_contourf(self, show=True, save=False):
        mi, mx = self.pss.min(), self.pss.max()
        lb1, ub1, lb2, ub2 = self.xishu_s.min(), self.xishu_s.max(), self.mici_s.min(), self.mici_s.max()
        print(mi, mx, lb1, ub1, lb2, ub2)
        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('比例系数')
        plt.ylabel('幂指数')
        plt.title("平均输出功率")
        if save:
            plt.savefig(IMG_SVG / '问题2-二维等高线图.svg')
        if show:
            plt.show()
        return True

    def _plotly_Contour(self, show=True, save=False):
        fig = go.Figure(data=go.Contour(
            z=self.pss,
#             x=self.xishu_s, y=self.mici_s,
#             fillcolor=True,
#             colorscale='jet',
            autocolorscale=False,
            autocontour=False,
            colorscale=[[0.0, "rgb(165,0,38)"],
                        [1/9, "rgb(215,48,39)"],
                        [2/9, "rgb(244,109,67)"],
                        [3/9, "rgb(253,174,97)"],
                        [4/9, "rgb(254,224,144)"],
                        [6/9, "rgb(224,243,248)"],
                        [6/9, "rgb(171,217,233)"],
                        [8/9, "rgb(116,173,209)"],
                        [8/9, "rgb(69,117,180)"],
                        [1.0, "rgb(49,54,149)"]],
        ))

        if save:
            fig.write_image('temp.png')
        if show:
            fig.show()
        return True

    def _plotly_Surface(self, show=True, save=False):
        fig = go.Figure(data=[go.Surface(
            x=self.xishu_s,
            y=self.mici_s,
            z=self.pss,
        )])

        if save:
            fig.write_image(IMG_SVG / '问题2-3D曲线图.svg')
        if show:
            fig.show()
        return True

    def plot_result(self, *args, **kwargs):
        self._plt_contourf(*args, **kwargs)
        self._plotly_Surface(*args, **kwargs)
        return None
```

In [13]:

```
solver2 = solver_task2()
solver2.epoch = 100
solver2.pprint = False
```

```
solver2.jit = False

solver2.run_GA()  # 遗传
solver2.run_Program(x0=[42424, 0.5])  # 规划
solver2.run_LMS(xishu_lb=0, xishu_ub=100000, xishu_num=10, xishu_alpha=0,
                mici_lb=0, mici_ub=1, mici_num=10, mici_alpha=0)  # 变步长
# solver2.run_LMS(xishu_lb=0, xishu_ub=100000, xishu_num=1000000, xishu_alpha=0,
#                 mici_lb=0, mici_ub=1, mici_num=100, mici_alpha=0)  # 遍历：运行时间很长……

solver2.plot_result(show=True, save=False)  # 绘图
```

```
 遗传开始计时：
[比例系数，幂指数]：  [6.75209262e+04 2.57741169e-01] 最大输出功率 [227.60337422]
 遗传结束计时，用时：  11.672255277633667 s
```

```
Out[13]: (array([6.75209262e+04, 2.57741169e-01]), array([227.60337422]))
```

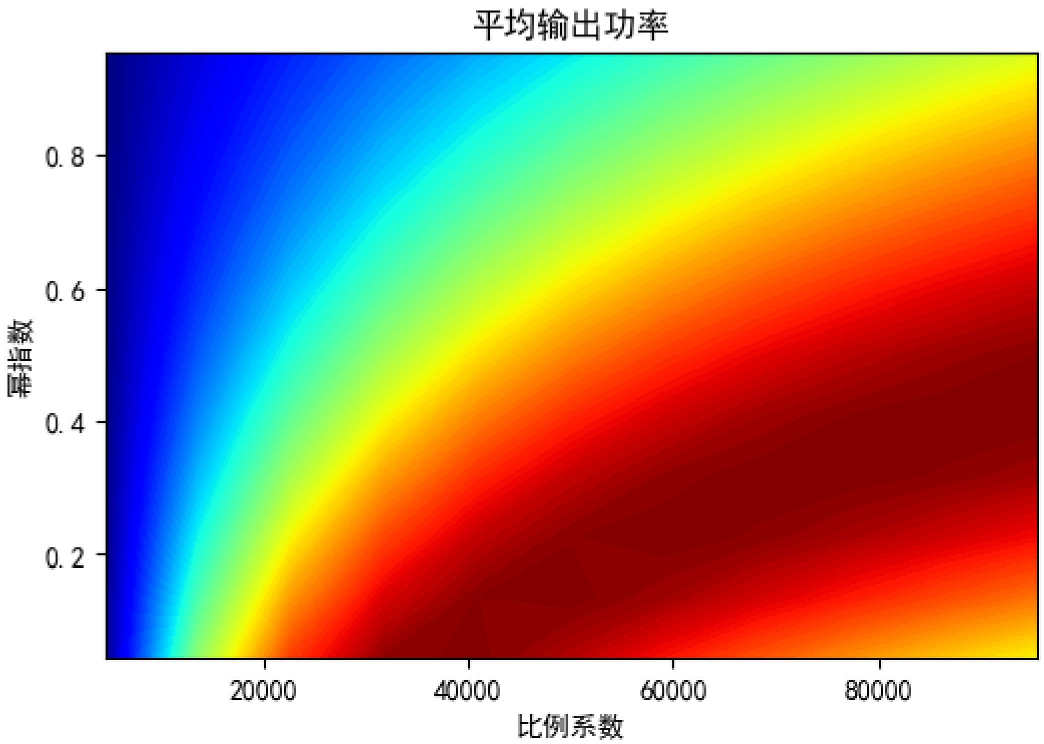
```
 规划开始计时：
Optimization terminated successfully    (Exit mode 0)
            Current function value: -225.16902175018322
            Iterations: 7
            Function evaluations: 71
            Gradient evaluations: 7
[比例系数，幂指数]：  [4.24237907e+04 8.42700465e-04] 最大输出功率 225.16902175018322
 规划结束计时，用时：  12.281152248382568 s
```

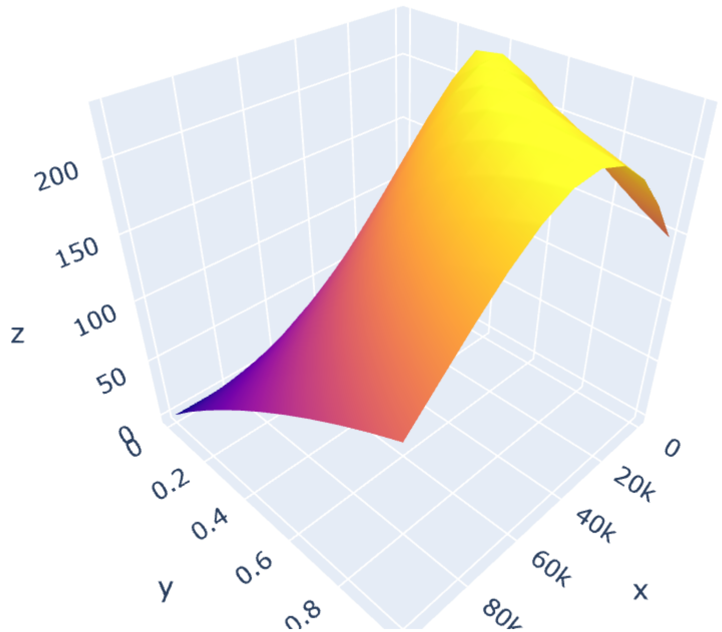
```
Out[13]: (array([4.24237907e+04, 8.42700465e-04]), 225.16902175018322)
```

```
 变步长开始计时：
 0/10 [..........] 1.75 s/iter
 1/10 [*.........] 1.97 s/iter
 2/10 [**........] 1.93 s/iter
 3/10 [***.......] 1.93 s/iter
 4/10 [****......] 2.04 s/iter
 5/10 [*****.....] 2.1 s/iter
 6/10 [******....] 2.49 s/iter
 7/10 [*******...] 2.17 s/iter
 8/10 [********..] 2.16 s/iter
 9/10 [*********.] 2.75 s/iter
10/10 [**********] 2.33 s/iter
[比例系数，幂指数]：  [100000.0, 0.4] 最大输出功率 227.64617885502085
 变步长结束计时，用时：  23.62815570831299 s
```

```
Out[13]: ([100000.0, 0.4], 227.64617885502085)
```

```
0.0 227.64617885502085 0.0 100000.0 0.0 1.0
```


平均输出功率
幂指数
0.8
0.6
0.4
0.2
20000
40000
60000
80000
比例系数



In [14]:

```
# # TODO 手动检查数据
# get_power(task=2, pprint=False, xishu=3.71721006e+04, mici=7.90606784e-02)
# get_power(task=2, pprint=False, xishu=4.24228511e+04, mici=5.01296092e-17)
# get_power(task=2, pprint=False, xishu=100000, mici=0.4)
```

In [ ]:

In [15]:

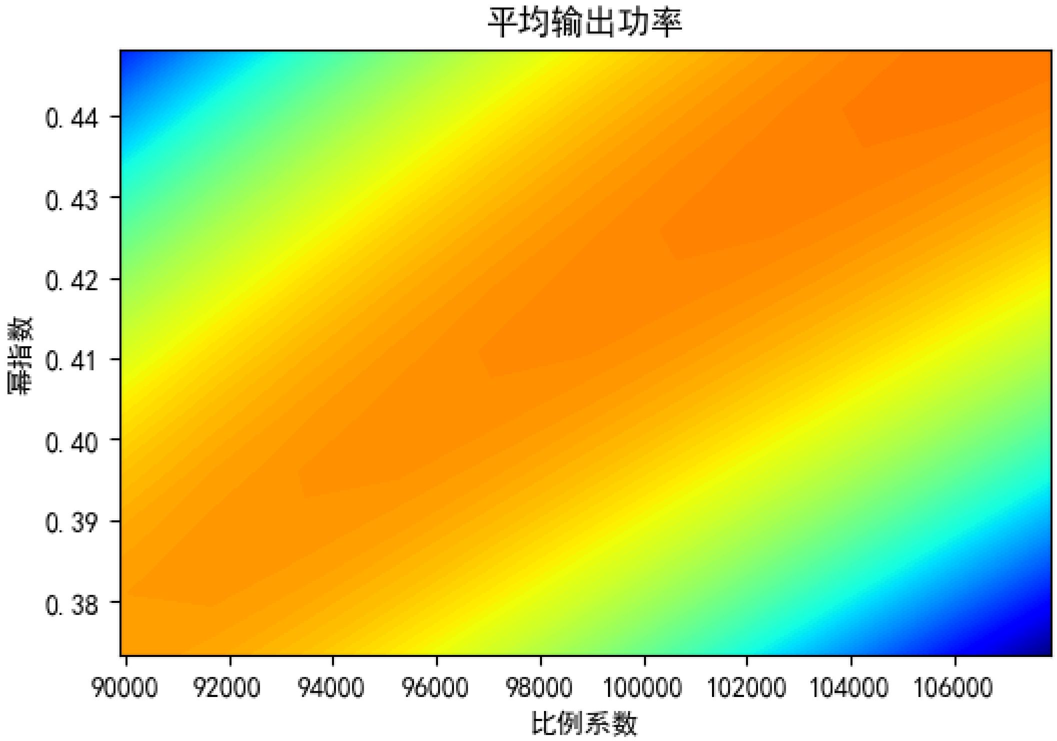
```
# TODO 联合运行求解
solver2 = solver_task2()

# TODO GA + LMS
solver2.epoch = 10
```

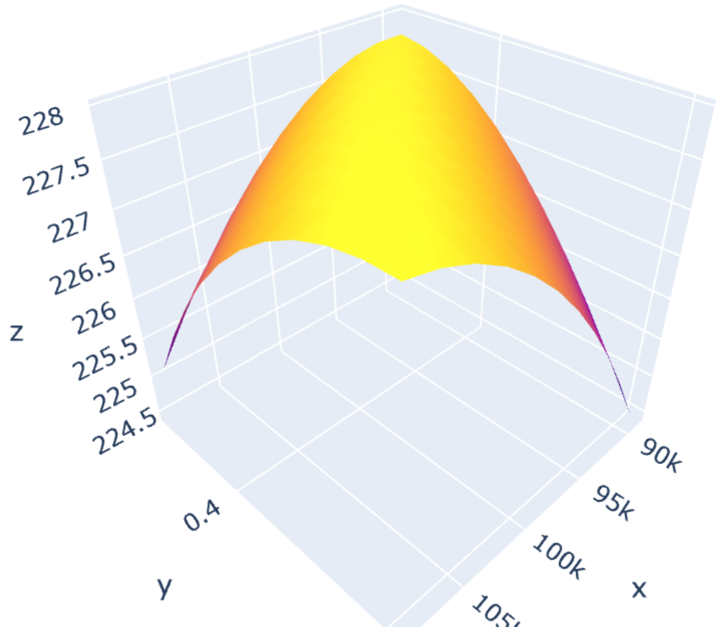
```python
solver2.pprint = False
solver2.jit = False
t0 = time()
xbest, ybest = solver2.run_GA()  # 遗传
# xbest = (80000, 1)
solver2.run_LMS(xishu_lb=xbest[0], xishu_ub=xbest[0], xishu_num=10, xishu_alpha=0.1,
                mici_lb=xbest[1], mici_ub=xbest[1], mici_num=10, mici_alpha=0.1)  # 变步长
print("总用时: ", time() - t0)
solver2.plot_result(show=True, save=False)  # 绘图

print('-' * 100)

# TODO Program + LMS
solver2.epoch = 100
solver2.pprint = False
solver2.jit = False
t0 = time()
xbest, ybest = solver2.run_Program(x0=[80000, 0.4])  # 规划
solver2.run_LMS(xishu_lb=xbest[0], xishu_ub=xbest[0], xishu_num=10, xishu_alpha=0.1,
                mici_lb=xbest[1], mici_ub=xbest[1], mici_num=10, mici_alpha=0.1)  # 变步长
print("总用时: ", time() - t0)
solver2.plot_result(show=True, save=False)  # 绘图
```

```
 遗传开始计时：
[比例系数, 幂指数]:  [9.88658007e+04 4.10816217e-01] 最大输出功率 [227.84305425]
 遗传结束计时，用时：  10.767147541046143 s
 变步长开始计时：
 0/10 [..........] 2.34 s/iter
 1/10 [*.........] 2.49 s/iter
 2/10 [**........] 2.54 s/iter
 3/10 [***.......] 2.56 s/iter
 4/10 [****......] 2.47 s/iter
 5/10 [*****.....] 2.78 s/iter
 6/10 [******....] 2.49 s/iter
 7/10 [*******...] 2.65 s/iter
 8/10 [********..] 2.47 s/iter
 9/10 [*********.] 2.57 s/iter
10/10 [**********] 2.41 s/iter
[比例系数, 幂指数]:  [108752.3808184858, 0.4518978388248586] 最大输出功率 227.94186216235866
 变步长结束计时，用时：  27.748749017715454 s
```

Out[15]:
```
([108752.3808184858, 0.4518978388248586], 227.94186216235866)
```

```
总用时：  38.5202112197876
224.47140601527843 227.94186216235866 88979.22066967019 108752.3808184858 0.36973459540215703 0.4518978388248586
```

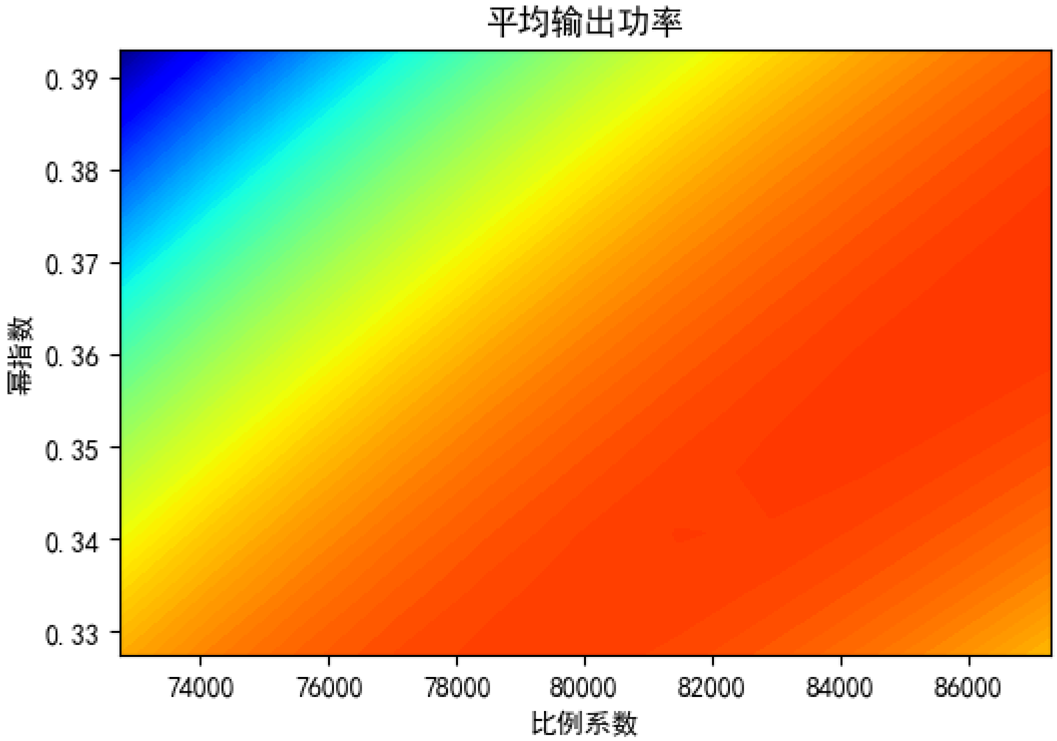

平均输出功率
幂指数
0.44
0.43
0.42
0.41
0.40
0.39
0.38
90000
92000
94000
96000
98000
100000
102000
104000
106000
比例系数

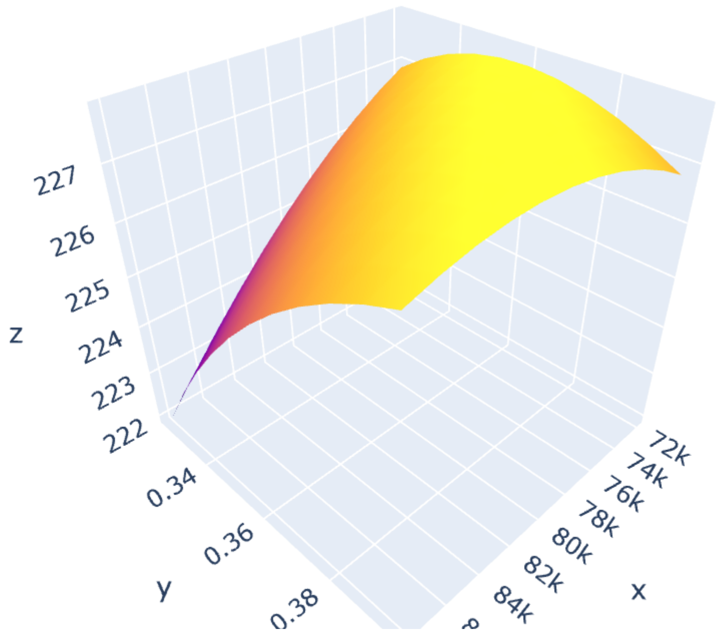

228
227.5
227
226.5
226
225.5
225
224.5
z
0.4
y
90k
95k
100k
105k
x

```
----------------------------------------------------------------------------------------------------
 规划开始计时：
Optimization terminated successfully    (Exit mode 0)
            Current function value: -227.24712185040116
            Iterations: 10
            Function evaluations: 117
            Gradient evaluations: 10
[比例系数, 幂指数]:  [8.00026845e+04 3.60299680e-01] 最大输出功率 227.24712185040116
 规划结束计时, 用时:  24.791813850402832 s
 变步长开始计时：
 0/10 [..........] 2.4 s/iter
 1/10 [*.........] 2.38 s/iter
 2/10 [**........] 2.08 s/iter
 3/10 [***.......] 2.29 s/iter
 4/10 [****......] 2.31 s/iter
 5/10 [*****.....] 2.65 s/iter
 6/10 [******....] 2.36 s/iter
 7/10 [*******...] 2.31 s/iter
 8/10 [********..] 2.39 s/iter
 9/10 [*********.] 2.46 s/iter
10/10 [**********] 2.37 s/iter
[比例系数, 幂指数]:  [88002.9529980827, 0.3675056738229399] 最大输出功率 227.7653029616472
 变步长结束计时, 用时:  26.013054847717285 s
```

```
Out[15]: ([88002.9529980827, 0.3675056738229399], 227.7653029616472)
```

```
总用时:  50.80678367614746
222.00697113533874 227.7653029616472 72002.41608934039 88002.9529980827 0.3242697121967117 0.3963296482404254
```


平均输出功率
幂指数
0.39
0.38
0.37
0.36
0.35
0.34
0.33
74000
76000
78000
80000
82000
84000
86000
比例系数



```
In [16]: # TODO 手动检查数据
get_power(task=2, pprint=False, xishu=59552.85847047485, mici=0.20415461684194905)
get_power(task=2, pprint=False, xishu=88002.9529980827, mici=0.3675056738229399)
```

```
Out[16]: 227.55871280599794

Out[16]: 227.7653029616472
```

```
In [ ]:
```