# 问题1

计算浮子和振子在波浪激励力 $fcos\omega t$（$f$ 为波浪激励力振幅，$\omega$ 为波浪频率）作用下前 $40$ 个波浪周期内时间间隔为 $0.2\ s$ 的垂荡位移和速度：

(1) 直线阻尼器的阻尼系数为 $10000\ N\cdot s/m$;
(2) 直线阻尼器的阻尼系数与浮子和振子的相对速度的绝对值的幂成正比，其中比例系数取 $10000$，幂指数取 $0.5$。

将结果存放在 result1-1.xlsx 和result1-2.xlsx 中。在论文中给出 $10s$、$20s$、$40s$、$60s$、$100s$ 时，浮子与振子的垂荡位移和速度。

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

import time
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
learn.metrics import mean_absolute_error as MAE
```

```python
from sklearn.metrics import classification_report, roc_auc_score

import sko
from sko.GA import GA

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")
```



```python
DIR
```

```python
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 [3]:
```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 [4]:
```python
# 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
```

Typesetting math: 100%

```python
            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(1)
_ = trange(0, 200, 0.2)
```

In [5]:

```
# 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)
    """
```

```python
    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
```

# 浮子和振子的微分方程



子位移，$X_2(t)$ 振子位移

$$m_2\frac{d^2X_2(t)}{dt^2}+c\frac{dX_2(t)}{dt}+kX_2(t)=c\frac{dX_1(t)}{dt}+kX_1(t)$$
$$m_1'\frac{d^2X_1(t)}{dt^2}+m_2\frac{d^2X_2(t)}{dt^2}=F(t)$$

其中，

$m_1$ 为浮子质量，单位 $kg$

$m_e$ 为垂荡附加质量，单位 $kg$

$m_1'=m_1+m_{垂荡附加质量}$

$m_2$ 为振子质量，单位 $kg$

$F(t)$ 为浮子和振子的合外力/垂向波浪力，单位 $N$

$k$ 为弹簧的劲度系数，单位 $N/m$

$c$ 为阻尼器的阻尼系数

$F(t)=F_{波浪激励力}-F_{兴波阻尼力}-F_{静水恢复力}+F_{浮力}-F_{重力}$

$=fcos(\omega t)-k_{垂荡兴波阻尼系数}v-\rho gSh+0$

$=fcos(\omega t)-k_{垂荡兴波阻尼系数}\frac{dX1(t)}{dt}-\rho gSX_1(t)$

$=fcos(\omega t)+k_1\frac{dX1(t)}{dt}+k_2X_1(t)$

其中，

$k_1=-k_{垂荡兴波阻尼系数}$

$k_2=-\rho gS$

# 微分方程的解析解（没前途）

In [6]:
```
# X1, X2 = sympy.symbols("X1 X2")
m1, m2, F, k, c = sympy.symbols("m1 m2 F k c")
```

Typesetting math: 100%

In [7]:

```python
# # （没前途）sympy 解微分方程的解析解：X1(t)、X2(t)

# m1, m2, k, c = sympy.symbols("m1 m2 k c")

# t = sympy.symbols('t')
# X1 = sympy.symbols('X1', cls=sympy.Function)
# X2 = sympy.symbols('X2', cls=sympy.Function)
# F = sympy.symbols('F', cls=sympy.Function)

# eq1 = sympy.Eq(
#     m2 * X2(t).diff(t, 2) + c * X2(t).diff(t, 1) + k * X2(t),
#     c * X1(t).diff(t, 1) + k * X1(t))
# eq2 = sympy.Eq(m1 * X1(t).diff(t, 2) + m2 * X2(t).diff(t, 2), F(t))
# eq = (eq1, eq2)

# t0 = time.time()
# res = sympy.dsolve(eq)
# print("用时：", time.time() - t0)

# print(sympy.latex(res))
```

In [8]:

```python
# # （没前途）sympy 解微分方程的解析解：X1(t)、X2(t)

# m1, m2, F, k, c = sympy.symbols("m1 m2 F k c")
# f, omega, k兴, k系, k重 = sympy.symbols("f omega k兴 k系 k重")

# t = sympy.symbols('t')
# X1 = sympy.symbols('X1', cls=sympy.Function)
# X2 = sympy.symbols('X2', cls=sympy.Function)

# eq1 = sympy.Eq(
#     m2 * X2(t).diff(t, 2) + c * X2(t).diff(t, 1) + k * X2(t).diff(t, 0),
#     c * X1(t).diff(t, 1) + k * X1(t).diff(t, 0)
# )
# eq2 = sympy.Eq(
#     m1 * X1(t).diff(t, 2) + m2 * X2(t).diff(t, 2),
#     f * sympy.cos(omega * t) + k兴 * X2(t).diff(t, 1) + k系 * X2(t).diff(t, 0) - k重
# )
# eq = (eq1, eq2)

# t0 = time.time()
# res = sympy.dsolve(eq)
# print("用时：", time.time() - t0)

(sympy.latex(res))
```

Typesetting math: 100%

In [9]:

```
# # （没前途）sympy 解微分方程的解析解：X1(t)、X2(t)

# TODO step1：求解析解
# m1, m2, F, k, c = sympy.symbols("m1 m2 F k c")
# f, omega, k兴, k系, k重 = sympy.symbols("f omega k兴 k系 k重")

# m1 = 浮子质量
# m2 = 振子质量
# 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
# k兴 = 垂荡兴波阻尼系数
# k系 = 海水的密度 * 重力加速度 * S浮子底面积
# k重 = F附加惯性力 + F重力

# t = sympy.symbols('t')
# X1 = sympy.symbols('X1', cls=sympy.Function)
# X2 = sympy.symbols('X2', cls=sympy.Function)

# eq1 = sympy.Eq(
#     m2 * X2(t).diff(t, 2) + c * X2(t).diff(t, 1) + k * X2(t).diff(t, 0),
#     c * X1(t).diff(t, 1) + k * X1(t).diff(t, 0)
# )
# eq2 = sympy.Eq(
#     m1 * X1(t).diff(t, 2) + m2 * X2(t).diff(t, 2),
#     f * sympy.cos(omega * t) + k兴 * X2(t).diff(t, 1) + k系 * X2(t).diff(t, 0) - k重
# )
# eq = (eq1, eq2)


# t0 = time.time()
# res = sympy.dsolve(eq)
# print("用时：", time.time() - t0)

# print(sympy.latex(res))

# TODO step1：带入具体数值求解
# # F = (F静水恢复力 + F兴波阻尼力 + F波浪激励力) - (F附加惯性力 + F重力)
# c直线阻尼器的阻尼系数 = c直线阻尼器的阻尼系数_func1()
# mapping = [
#     (m1, 浮子质量),
#     (m2, 振子质量),
#     (k, 弹簧刚度),
#     (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),
#     (k兴, 垂荡兴波阻尼系数),
#     (k系, 海水的密度 * 重力加速度 * S浮子底面积),
#     (k重, F附加惯性力 + F重力),
# ]
# res.subs(mapping)
```

# 微分方程的数值解（可行）

$F(t) = F_{\text{波浪激励力}} - F_{\text{兴波阻尼力}} - F_{\text{静水恢复力}}$

$= fcos(\omega t) - k_{\text{垂荡兴波阻尼系数}} v - \rho g S h$

$= fcos(\omega t) - k_{\text{垂荡兴波阻尼系数}} \frac{dX1(t)}{dt} - \rho g S X_1(t)$

$= fcos(\omega t) + k_1 \frac{dX1(t)}{dt} + k_2 X_1(t)$

其中，

$k_1 = -k_{\text{垂荡兴波阻尼系数}}$

$k_2 = -\rho g S$

为解上述微分方程的数值解，求出：

$$y' = \begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix} = \begin{bmatrix} \dot{X}_1 \\ \ddot{X}_1 \\ \dot{X}_2 \\ \ddot{X}_2 \end{bmatrix} = \begin{bmatrix} \dot{X}_1 \\ (fcos(\omega t) + k_1 \dot{X}_1 + k_2 X_1 - m_2 \ddot{X}_2)/m_1' \\ \dot{X}_2 \\ (c(\dot{X}_1 - \dot{X}_2) + k(X_1 - X_2))/m_2 \end{bmatrix} = \begin{bmatrix} y_2 \\ (fcos(\omega t) + k_1 y_2 + k_2 y_1 - m_2 \ddot{X}_2)/m_1' \\ y_4 \\ (c(y_2 - y_4) + k(y_1 - y_3))/m_2 \end{bmatrix}$$

分别求解一下两种情况的微分方程的数值解：

（1）$c = 10000$

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

# 准备



定义常量、画图函数等

In [13]:

```
# 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
k2 = -海水的密度 * 重力加速度 * S浮子底面积

def plot_mat(t, y):
    """ matplotlib 画图函数，画图效果一般，未使用 """
    plt.figure(dpi=100)
    plt.plot(t, y[:, 0], label="浮子位移" + "$X1$")
    plt.plot(t, y[:, 1], label="浮子速度" + "$V1$")
    plt.plot(t, y[:, 2], label="振子位移" + "$X2$")
    plt.plot(t, y[:, 3], label="振子速度" + "$V2$")
    plt.legend()
#     plt.grid()
    plt.show()
    return None

def plot_plotly(t, y, title, svg_name=None):
    """ plotly 画图函数
    :param t: 时间
    :param y: 浮子、振子的垂荡位移、速度
    :param title: 绘图标题
    :param svg_name: 保存成矢量图的文件名称
    """
    trace1 = go.Scatter(x=t, y=y[:, 0], name="$浮子垂荡位移X_1$", yaxis='y1')
    trace2 = go.Scatter(x=t, y=y[:, 1], name="$浮子垂荡速度V_1$", yaxis='y2')
    trace3 = go.Scatter(x=t, y=y[:, 2], name="$振子垂荡位移X_2$", yaxis='y1')
    trace4 = go.Scatter(x=t, y=y[:, 3], name="$振子垂荡速度V_2$", yaxis='y2')
#     trace1 = go.Scatter(x=t, y=y[:, 0], name="$浮子垂荡位移X_1$", line={"width": 1})
#     trace2 = go.Scatter(x=t, y=y[:, 1], name="$浮子垂荡速度V_1$", line={"width": 1})
#     trace3 = go.Scatter(x=t, y=y[:, 2], name="$振子垂荡位移X_2$", line={"width": 1})
#     trace4 = go.Scatter(x=t, y=y[:, 3], name="$振子垂荡速度V_2$", line={"width": 1})
#     fig = go.Figure(data=[trace1, trace3])
    fig = go.Figure(data=[trace1, trace2, trace3, trace4])
    fig.update_layout(
        width=1000,
        height=600,
```

Typesetting math: 100%

```python
        xaxis=dict(title='$时间 (s)$'),
        yaxis=dict(title='$垂荡位移 (m)$'),
        yaxis2=dict(title='$垂荡速度 (m/s)$', anchor='x', overlaying='y', side='right'),
        legend=dict(y=1.22, yanchor="top", x=1, xanchor="right"),
        title=title,
    )
    if svg_name is not None:
        fig.write_image(IMG_SVG / svg_name)
    fig.show()
    return None

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, 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=10000, a=0.5)
    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_result1_df(result1, t):
    """
    :param result1: np.ndarray 问题1的结果（有两小问）
    :param t: np.ndarray 时间
    """
    columns = ['时间 (s)', '浮子位移 (m)', '浮子速度 (m/s)', '振子位移 (m)', '振子速度 (m/s)']
    shijian = pd.DataFrame(t, columns=columns[:1])
    result1 = pd.DataFrame(result1, columns=columns[1:])
    result1_df = pd.concat([shijian, result1], axis=1)
    return result1_df
```

```python
def save_result1_df(task, result1_df, save=True):
    # 前 40 周期，间隔 0.2s
    result1_1 = result1_df.iloc[:int(40 * 波浪周期 / t_step) + 1, :]
    if save:
        file_name = 'result1-1.csv' if task == 1 else 'result1-2.csv'
        result1_1.to_csv(file_name, encoding='utf_8_sig')  # 附件中的结果

    #  10 s、20 s、40 s、60 s、100 s
    idx = list(map(lambda x: x * 5, [10, 20, 40, 60, 100]))
    result1_1_paper = result1_1_df.iloc[idx, :]
    if save:
        file_name = 'result1-1-paper.csv' if task == 1 else 'result1-2-paper.csv'
        result1_1_paper.to_csv(file_name, encoding='utf_8_sig')  # 论文中的结果

    print("已保存结果！")
    return None

def get_power(task,
              t_left=0,
              t_right=100,
              t_step=None,
              c=c, k=k, k1=k1, k2=k2,
              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
    :param t_step: 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=tuple())
    elif task == 2:
        result1 = odeint(differential_equations_2, y0, t, args=tuple())

    delta = abs(result1[:, 1] - result1[:, 3])  # 相对速度
    if task == 1:
        c = 10000
    elif task == 2:
        c = 10000 * delta**0.5
    f_zuli = c * delta
```

```python
    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

    stable_time_begin = 60
    stable_time_end = 100
    stable_time_length = stable_time_end - stable_time_begin

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

    power = power_i[idx_begin:idx_end]
    P = power.sum() / stable_time_length
    return P

def plot_fzj_szj(task, title,
#                  t_left, t_right, t_step,
                 svg_name=None,
                 y0=[0, 0, 0, 0]):
    """ 绘制仿真解和数值解的图像（该函数与其他函数独立）
    :param t_left: 设置为 0, 固定值
    :param t_right: 设置为 200, 固定值
    :param t_step: 设置为 0.2, 固定值
    """
    # 稳定的时间区间
    t_step = 0.2
    t_left, t_right = 0, 200

    stable_time_begin = 70
    stable_time_end = 140

    # 仿真数据（队友在 MATLAB 上的运行结果）
    data_fangzhenjie_path = 'data500.xlsx' if task == 1 else 'data500-diff_c.xlsx'
    data_fangzhenjie = pd.read_excel(DATA_COOKED / data_fangzhenjie_path)
    data_fangzhenjie

    cond1 = abs(data_fangzhenjie.iloc[:, 0] - stable_time_begin) < 0.1
    idx_min = data_fangzhenjie[cond1].index[0]
    cond2 = abs(data_fangzhenjie.iloc[:, 0] - stable_time_end) < 0.1
    idx_max = data_fangzhenjie[cond2].index[0]

    plot_data_fangzhenjie_time = data_fangzhenjie.iloc[idx_min: idx_max, 0]
    plot_data_fangzhenjie = data_fangzhenjie.iloc[idx_min: idx_max, 1]

    # 微分方程数值解
```

```python
    t = np.linspace(t_left, t_right, num=int(t_right / t_step) + 1)
    differential_equations = differential_equations_1 if task == 1 else differential_equations_2
    data_shuzhijie = odeint(differential_equations, y0, t)

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

    plot_data_shuzhijie_time = t[idx_begin: idx_end+1]
    plot_data_shuzhijie = data_shuzhijie[idx_begin: idx_end+1, 0] - data_shuzhijie[idx_begin: idx_end+1, 2]

    # 画图
    trace1 = go.Scatter(
        x=plot_data_fangzhenjie_time, y=plot_data_fangzhenjie,
        name='$仿真解$',
    )
    trace2 = go.Scatter(
        x=plot_data_shuzhijie_time, y=plot_data_shuzhijie,
        name='$数值解$',
    )

    fig = go.Figure(data=[trace1, trace2])
    fig.update_layout(
        width=1000,
        height=600,
        xaxis=dict(title='$时间 (s)$'),
        yaxis=dict(title='$相对垂荡位移 (m)$'),
        legend=dict(y=1.13, yanchor="top", x=1, xanchor="right"),
        title=title,
    )
    if svg_name is not None:
        fig.write_image(IMG_SVG / svg_name)
    fig.show()
    del fig
    return None
```

（1） c 常数

1. 求解浮子和振子的垂荡位移和垂荡速度
2. 保存结果
3. 绘图
4. 对比数值解和仿真解
5. 观察相对位移、相对速度
6. 查看平均输出功率



$c = 10000$

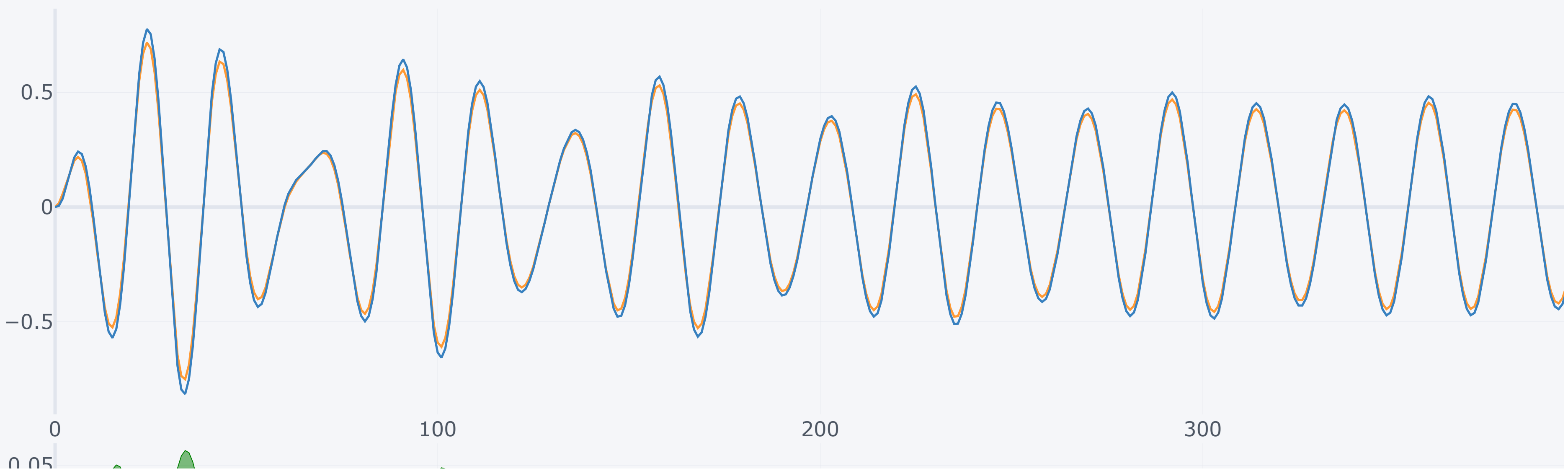
In [14]:

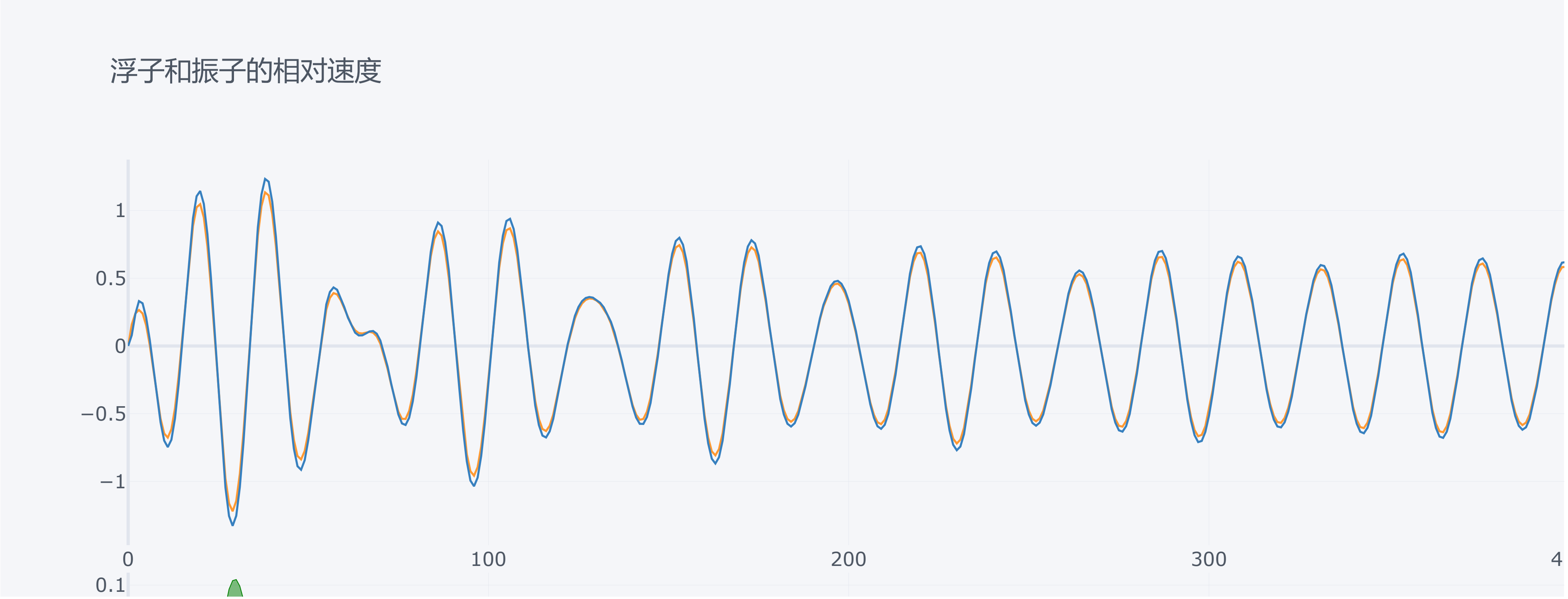
```
# TODO 设置参数，求解
_ = trange(0, 100, 0.2)
t = np.linspace(t_left, t_right, num=int(t_right / t_step) + 1)
y0 = [0, 0, 0, 0]
result1_1 = odeint(differential_equations_1, y0, t)

# TODO 保存结果
result1_1_df = get_result1_df(result1_1, t)
save_result1_df(task=1, result1_df=result1_1_df, save=True)

# TODO 绘图（第 1 个图）
# plot_mat(t, result1_1)
plot_plotly(t=t, y=result1_1,
            title='$直线阻尼器的阻尼系数为常数——浮子和振子的垂荡位移和速度$',
            svg_name="问题1-c常数：浮子振子位移速度图.svg")

# TODO 对比：仿真结果、数值结果（第 2 个图）
plot_fzj_szj(task=1,
             title="$直线阻尼器的阻尼系数为常数——浮子和振子的相对垂荡位移$",
             svg_name="问题1-c常数：浮子振子相对位移图.svg")

# TODO 准备问题2：观察相对位移、相对速度（第 3、4 个图）
result1_1_df.iloc[:, [1, 3]].iplot(kind='spread', title='浮子和振子的相对位移')
result1_1_df.iloc[:, [2, 4]].iplot(kind='spread', title='浮子和振子的相对速度')

# TODO 查看平均输出功率（从问题2回来看一下问题1的功率）
print()
print("平均输出功率：", get_power(task=1, t_step=0.01), "W")
print()
```

已保存结果！

Typesetting math: 100%

浮子和振子的相对位移






平均输出功率： 240.5413505711107 W

（2）c 非常数

1. 求解浮子和振子的垂荡位移和垂荡速度
2. 保存结果
3. 绘图
4. 对比数值解和仿真解
5. 观察相对位移、相对速度
6. 查看平均输出功率

Typesetting math: 100%

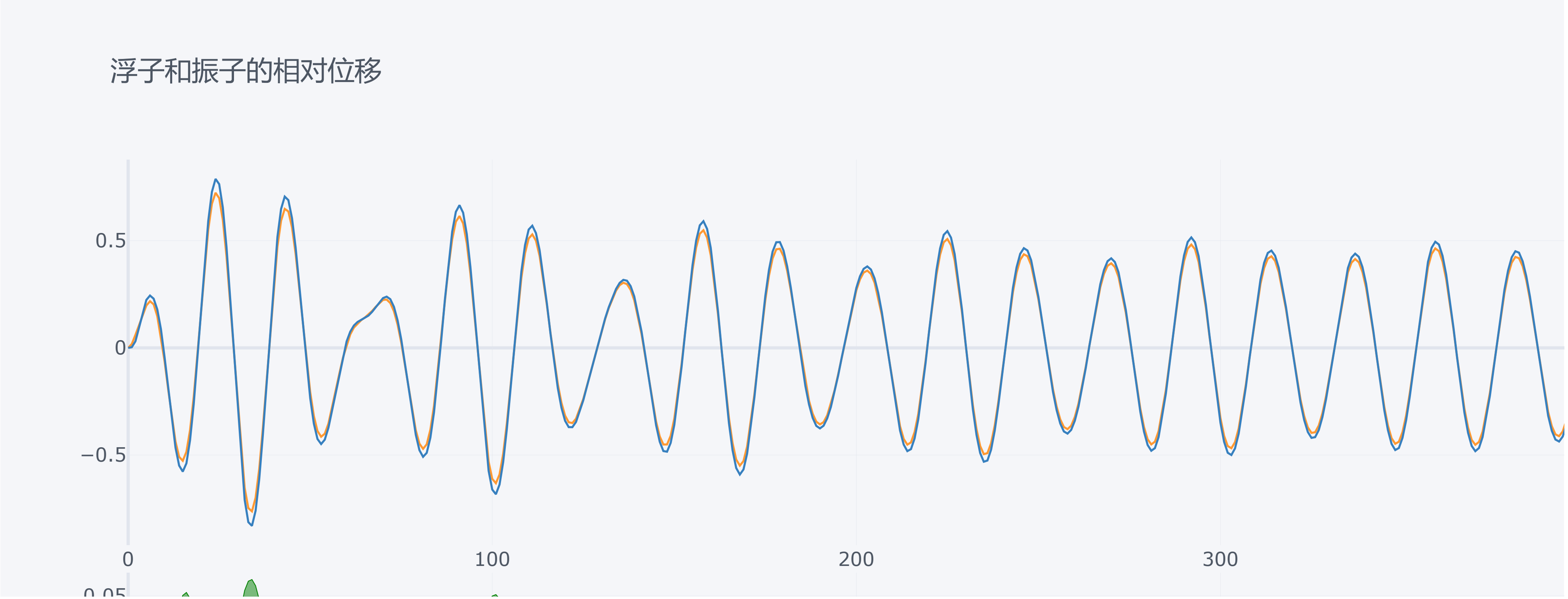
$c = 10000|V|^{0.5} = 10000|V_1 - V_2|^{0.5} = 10000|\dot{X}_1 - \dot{X}_2|^{0.5}$

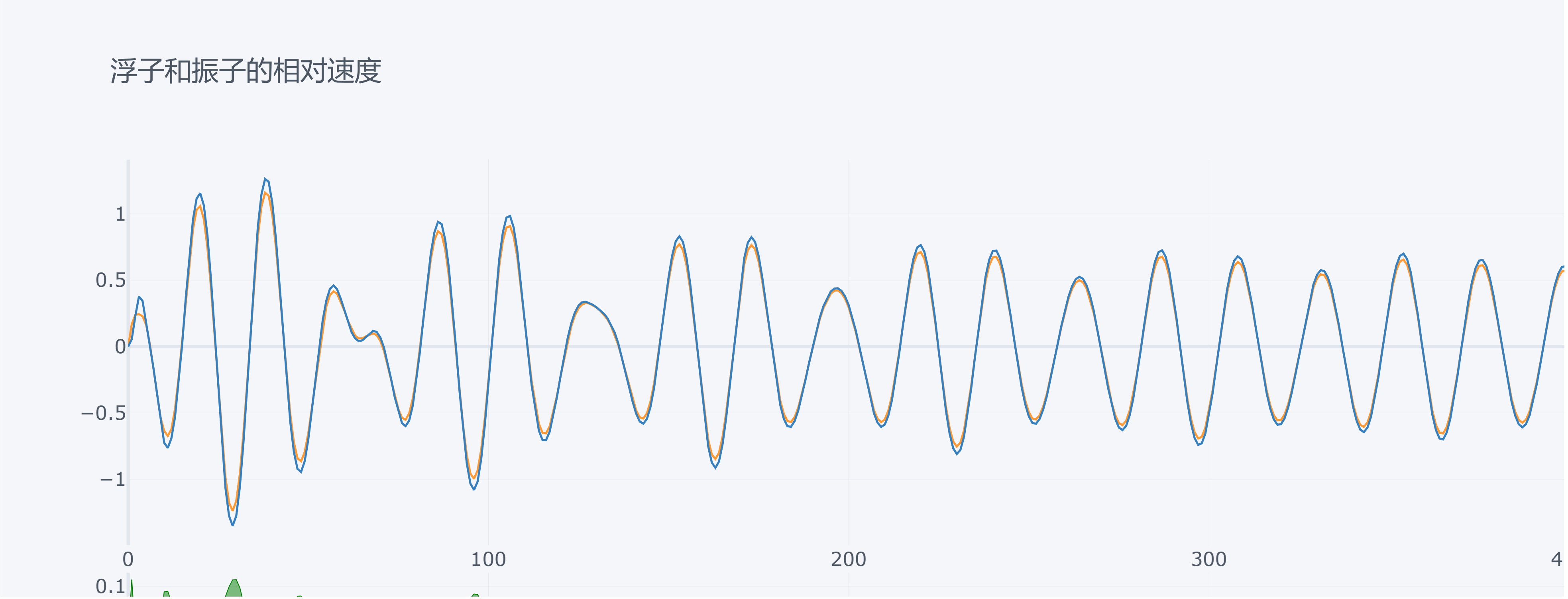
In [15]:

```
# TODO 数值解
_ = trange(0, 100, 0.2)
t = np.linspace(t_left, t_right, num=int(t_right / t_step) + 1)
y0 = [0, 0, 0, 0]
result1_2 = odeint(differential_equations_2, y0, t)

# TODO 保存结果
result1_2_df = get_result1_df(result1_2, t)
save_result1_df(task=2, result1_df=result1_2_df, save=True)

# TODO 绘图（第 1 个图）
# plot_mat(t, result1_2)
plot_plotly(t, result1_2,
            title='$直线阻尼器的阻尼系数为非常数——浮子和振子的垂荡位移和速度$',
            svg_name="问题1-c非常数：浮子振子位移速度图.svg")

# TODO 对比：仿真结果、数值结果（第 2 个图）
plot_fzj_szj(task=2,
             title="$直线阻尼器的阻尼系数为非常数——浮子和振子的相对垂荡位移$",
             svg_name="问题1-c非常数：浮子振子相对位移图.svg")

# TODO 准备问题2：观察相对位移、相对速度（第 3、4 个图）
result1_2_df.iloc[:, [1, 3]].iplot(kind='spread', title='浮子和振子的相对位移')
result1_2_df.iloc[:, [2, 4]].iplot(kind='spread', title='浮子和振子的相对速度')

# TODO 查看平均输出功率（从问题2回来看一下问题1的功率）
print()
print("平均输出功率：", get_power(task=2, t_step=0.01), "W")
print()
```

已保存结果！

Typesetting math: 100%


浮子和振子的相对位移
0.5
0
−0.5
0
100
200
300




平均输出功率： 42.43309470824312 W

In [ ]:

In [ ]:

In [ ]:

Typesetting math: 100%