Отчёт по практическому заданию.
6 семестр

Козлов Кирилл 305 группа

5 мая 2025 г.

Содержание

1 Постановка задачи.

- Требуется решить систему НДС:

$$\begin{cases} \dot{y}_1 = y_1 - \dfrac{D}{2} y_2 + y_2\left(y_3 + y_1^2\right) \\ \dot{y}_2 = \dfrac{D}{2} y_1 + y_2 + y_1\left(3y_3 - \dot{y}_1^2\right) \\ \dot{y}_3 = -2y_3\left(M + y_1 y_2\right) \end{cases}$$

  где D,M - константы ,положительные вещественные (M>1)

- Построить график решения и графиков зависимости y_1, y_2, y_3 от времени t.
- Узнать при каких значениях D,M у y_1, y_2, y_3 наблюдаются:

  1. Периодические режимы
  2. Хаотические режимы
  3. Режимы колебаний с ростом амплитуды или с затуханием

2 Аналитическое нахождение стационарной точки

Приравняем производные к 0, чтобы найти стационарное состояние:

$$\begin{cases} 0 = y_1 - \dfrac{D}{2} y_2 + y_2\left(y_3 + y_1^2\right) \\ 0 = \dfrac{D}{2} y_1 + y_2 + y_1\left(3y_3 - \dot{y}_1^2\right) \\ 0 = -2y_3\left(M + y_1 y_2\right) \end{cases}$$

Решая систему получим, систему:

$$\begin{cases} y_2^* & = \pm\sqrt{DM \pm \sqrt{D^2M^2 - 4M^3 + 3M^2}} \\ y_1^* & = -\dfrac{M}{y_2^*} \\ y_3^* & = \dfrac{D}{2} + \dfrac{M(1-M)}{(y_2^*)^2} \end{cases}$$

Таким образом, получим 4 возможных варианта стационарной точки.
Начальное положение системы определяется сдвигом ε:

$$y_i(0) = y_i^* + \varepsilon$$

3 Численное решение задачи

3.1 Метод Рунге-Кутта 4 порядка

Рассмотрим задачу Коши для обыкновенного дифференциального уравнения первого порядка:

$$\begin{cases} y' = f(x, y), \\ y(x_0) = y_0. \end{cases}$$

Разобьём отрезок $[x_0, x_N]$ с шагом h, где $x_n = x_0 + nh$. Тогда приближённое значение решения $y_n \approx y(x_n)$ вычисляется по следующей формуле метода Рунге–Кутты четвёртого порядка:

$$
\begin{aligned}
k_1 &= f(x_n, y_n), \\
k_2 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1\right), \\
k_3 &= f\left(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2\right), \\
k_4 &= f(x_n + h, y_n + hk_3), \\
y_{n+1} &= y_n + \frac{h}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right).
\end{aligned}
$$

Метод Рунге–Кутты 4-го порядка обладает локальной погрешностью порядка $\mathcal{O}(h^5)$ и глобальной погрешностью порядка $\mathcal{O}(h^4)$.

3.2 Таблица Бутчера для метода Рунге-Кутта 4 порядка

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Это таблица Бутчера для классического метода Рунге–Кутты 4-го порядка. В ней:
- Столбец слева (c_i) — точки, в которых вычисляется правая часть
- Центральная матрица (a_{ij}) — коэффициенты для промежуточных значений
- Нижняя строка (b_i) — веса для итогового взвешенного среднего.

4 Программа решения задачи.

4.1 Алгоритм решения.

Краткий алгоритм решения:

1. Вычисление значений функций и сохранение их
2. Построение графиков
3. Вариация постоянных

4.2 Программа решения

4.2.1 Вычисления

```
import numpy as np
import pandas as pd
import os
import matplotlib.gridspec as gridspec
import matplotlib.pyplot as plt
from ipywidgets import interact, IntSlider, Play, VBox, HBox, FloatSlider
import re
import math
import plotly.graph_objects as go
from IPython.display import display, clear_output
import ipywidgets as widgets


# удаляем старые данные
!rm -rf data_files/data_M*.csv
```

```python

# ==== Константы ====
# зафиксировать одну стационарную точку , один шаг eps = 0.5, один шаг
#                step=0.01 и D >= 2*sqrt(M)
T0 = 0.0
TN = 200.0
TIME_STEP = 0.5
EPS = 0.5

# ==== Класс состояния ====
class State:
    def __init__(self, y1, y2, y3):
        self.y1 = y1
        self.y2 = y2
        self.y3 = y3

    def copy(self):
        return State(self.y1, self.y2, self.y3)

# ==== Вычисления начальных условий ====
def compute_stationary_solutions(M, D, sign_1, sign_2):
    y_star = State(0, 0, 0)
    discriminant = D**2 * M**2 - 4*M**3 + 3*M**2

    try:
        inner_sqrt = D*M + sign_2*np.sqrt(discriminant)

        if inner_sqrt < 0:
            print(f"Warning: Подкоренное выражение отрицательное: D={D}, M={M},
                 inner_sqrt={inner_sqrt}")
            inner_sqrt = 0  # Чтобы избежать ошибки, подставляем 0

        y_star.y2 = sign_1*np.sqrt(inner_sqrt)

        # Защита от деления на ноль
        if y_star.y2 == 0:
            print(f"Warning: Деление на ноль при расчете y1: D={D}, M={M}")
            y_star.y1 = 0
        else:
            y_star.y1 = -M / y_star.y2

        # Аналогично защищаем y3
        if y_star.y2 == 0:
            y_star.y3 = 0
        else:
            y_star.y3 = D/2 + (M/(y_star.y2**2)) * (1 - M)

    except Exception as e:
        print(f"Ошибка при расчете стационарных решений: {e}")
        y_star = State(0, 0, 0)  # Если что-то совсем пошло не так, обнуляем
    return y_star

def compute_beginning_values(vector_y_star):
    return State(
        vector_y_star.y1 + EPS,
        vector_y_star.y2 + EPS,
        vector_y_star.y3 + EPS
    )
```

```python

    # ==== Правая часть системы ====
    def func(t, y, M, D):
        dy = State(0, 0, 0)

        dy.y1 = y.y1 - (D / 2.0) * y.y2 + y.y2 * (y.y3 + y.y1**2)
        dy.y2 = (D / 2.0) * y.y1 + y.y2 + y.y1 * (3 * y.y3 - y.y1**2)
        dy.y3 = -2.0 * y.y3 * (M + y.y1 * y.y2)
        return dy

    # ==== Метод Рунге-Кутты 4-го порядка ====
    def runge_kutta_4_system(M, D, sign_1, sign_2):
        vector_y_star = compute_stationary_solutions(M, D, sign_1, sign_2)
        vector_y_0 = compute_beginning_values(vector_y_star)
        t = T0
        y = vector_y_0
        results = []

        while (t <= TN + TIME_STEP):
            results.append((t, y.copy()))
            if (abs(y.y1) > 1e6 or abs(y.y2) > 1e6 or abs(y.y3) > 1e6):
            print(f"Warning: Решение ушло в бесконечность на t={t}")
            break

            k1 = func(t, y, M, D)
            k2 = func(t + TIME_STEP/2, State(
                y.y1 + TIME_STEP/2 * k1.y1,
                y.y2 + TIME_STEP/2 * k1.y2,
                y.y3 + TIME_STEP/2 * k1.y3
            ), M, D)
            k3 = func(t + TIME_STEP/2, State(
                y.y1 + TIME_STEP/2 * k2.y1,
                y.y2 + TIME_STEP/2 * k2.y2,
                y.y3 + TIME_STEP/2 * k2.y3
            ), M, D)
            k4 = func(t + TIME_STEP, State(
                y.y1 + TIME_STEP * k3.y1,
                y.y2 + TIME_STEP * k3.y2,
                y.y3 + TIME_STEP * k3.y3
            ), M, D)

            y.y1 += TIME_STEP/6 * (k1.y1 + 2*k2.y1 + 2*k3.y1 + k4.y1)
            y.y2 += TIME_STEP/6 * (k1.y2 + 2*k2.y2 + 2*k3.y2 + k4.y2)
            y.y3 += TIME_STEP/6 * (k1.y3 + 2*k2.y3 + 2*k3.y3 + k4.y3)

            t += TIME_STEP

        return results
```

4.2.2 Построение графиков

```python
def plot_trajectory(D,M,sign_1,sign_2):
    if (D >= np.sqrt(4*M-3)):
        results = runge_kutta_4_system(M,D,sign_1,sign_2)
        # Строим таблицу
        data = {
            'Time': [t for t, y in results],
```

```python
        'y1': [y.y1 for t, y in results],
        'y2': [y.y2 for t, y in results],
        'y3': [y.y3 for t, y in results],
    }

    df = pd.DataFrame(data)

    fig = plt.figure(figsize=(16, 8))
    gs = gridspec.GridSpec(3, 2, width_ratios=[2, 1]) # 3 rows ,2 columns
    ax = fig.add_subplot(gs[:, 0], projection='3d')
    # Starting and ending points
    start = df.iloc[0]
    end = df.iloc[-1]

    # Drow the start and end points
    ax.scatter([start["Time"]],[start["y1"]], [start["y2"]], [start["y3"]], color="green", s=15,
    ax.scatter([end["Time"]],[end["y1"]], [end["y2"]], [end["y3"]], color="red", s=15, label="En


    #  --- Projections start point ---
    # for y2-y3
    ax.plot([start["Time"],end["Time"]],[start["y1"], start["y1"]], [start["y2"], start["y2"]],
    ax.scatter([end["Time"]],[start["y1"]], [start["y2"]], [start["y3"]], color="green", s=5, zo
    # for y1-y2
    ax.plot([start["Time"],start["Time"]],[start["y1"], start["y1"]], [start["y2"], end["y2"]],
    ax.scatter([start["Time"]],[start["y1"]], [end["y2"]], [start["y3"]], color="green", s=5, zo
    # for y1-y3
    ax.plot([start["Time"],start["Time"]],[end["y1"], start["y1"]], [start["y2"], start["y2"]],
    ax.scatter([start["Time"]],[end["y1"]], [start["y2"]], [start["y3"]], color="green", s=5, zo

    # --- Projections end point ---
    # for  y2-y3
    ax.plot([end["Time"],start["Time"]],[end["y1"], end["y1"]], [end["y2"], end["y2"]], [start["
    ax.scatter([start["Time"]],[end["y1"]], [end["y2"]], [end["y3"]], color="red", s=5, zorder=5
    # for y1-y2
    ax.plot([end["Time"],end["Time"]],[end["y1"], end["y1"]], [end["y2"], start["y2"]], [end["y3
    ax.scatter([end["Time"]],[end["y1"]], [start["y2"]], [end["y3"]], color="red", s=5, zorder=5
    # for y1-y3
    ax.plot([end["Time"],end["Time"]],[start["y1"], end["y1"]], [end["y2"], end["y2"]], [end["y3
    ax.scatter([end["Time"]],[start["y1"]], [end["y2"]], [end["y3"]], color="red", s=5, zorder=5


    # Make trajectory for y1, y2, y3
    ax.plot(df["Time"],df["y1"], df["y2"], df["y3"], label="Trajectory", color="blue",linewidth=

    ax.set_xlabel("Y1", labelpad=10, fontsize=13,fontweight='bold')
    ax.set_ylabel("Y2", labelpad=10, fontsize=13,fontweight='bold')
    ax.set_zlabel("Y3", labelpad=10, fontsize=13,fontweight='bold')
    ax.set_title("График решения", fontsize=20)

    # Add legend and grid
    ax.legend(loc='best')
    ax.view_init(elev=40, azim=140)
    ax.grid(True)

    handles, labels = ax.get_legend_handles_labels()
    unique_labels = dict(zip(labels, handles))
    ax.legend(unique_labels.values(), unique_labels.keys())
```

```


        ax2 = fig.add_subplot(gs[0, 1])
        ax2.plot(df["Time"], df["y1"], color='blue')
        ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
        ax2.set_ylabel("y1".upper(), rotation=0, labelpad=13, fontsize=13,fontweight='bold')
        ax2.grid(True)

        ax2 = fig.add_subplot(gs[1, 1])
        ax2.plot(df["Time"], df["y2"], color='blue')
        ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
        ax2.set_ylabel("y2".upper(), rotation=0, labelpad=13,fontsize=13,fontweight='bold')
        ax2.grid(True)

        ax2 = fig.add_subplot(gs[2, 1])
        ax2.plot(df["Time"], df["y3"], color='blue')
        ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
        ax2.set_ylabel("y3".upper(), rotation=0, labelpad=13,fontsize=13,fontweight='bold')
        ax2.grid(True)

        #  Add text with constants to the left upper corner
        formula_str = (
            f'$y_2^* = {"" if sign_1 == 1 else "-"}\sqrt{{DM {"+ " if sign_2 == 1 else "- "} \sqrt{{
        )
        textstr = '\n'.join((
            r'$D=%s$' % D,
            r'$M=%s$' % M,
            r'$T_0=%.1f$' % T0,
            r'$T_N=%.1f$' % TN,
            r'$\varepsilon=%.1f$' % EPS,
            r'$STEP=%.1f$' % TIME_STEP,
            formula_str
        ))
        props = dict(boxstyle='round', facecolor='white', alpha=0.5)

        # Настройка отступов
        plt.subplots_adjust(left=0.05, right=0.95, top=0.95, bottom=0.1, hspace=0.4)
        plt.gcf().text(0.01, 0.8, textstr, fontsize=13, bbox=props)

        plt.show()
    else:
        print(f"Некоректные данные(D < sqrt(4*M-3)): {D} < sqrt(4*{M}-3)")
```

5 Графики состояний

Некоторые графики системы , при которых наблюдаются разные режимы у y_i. Хаотические режимы наблюдаются при любых $D > 2.0, M > 2.0$.

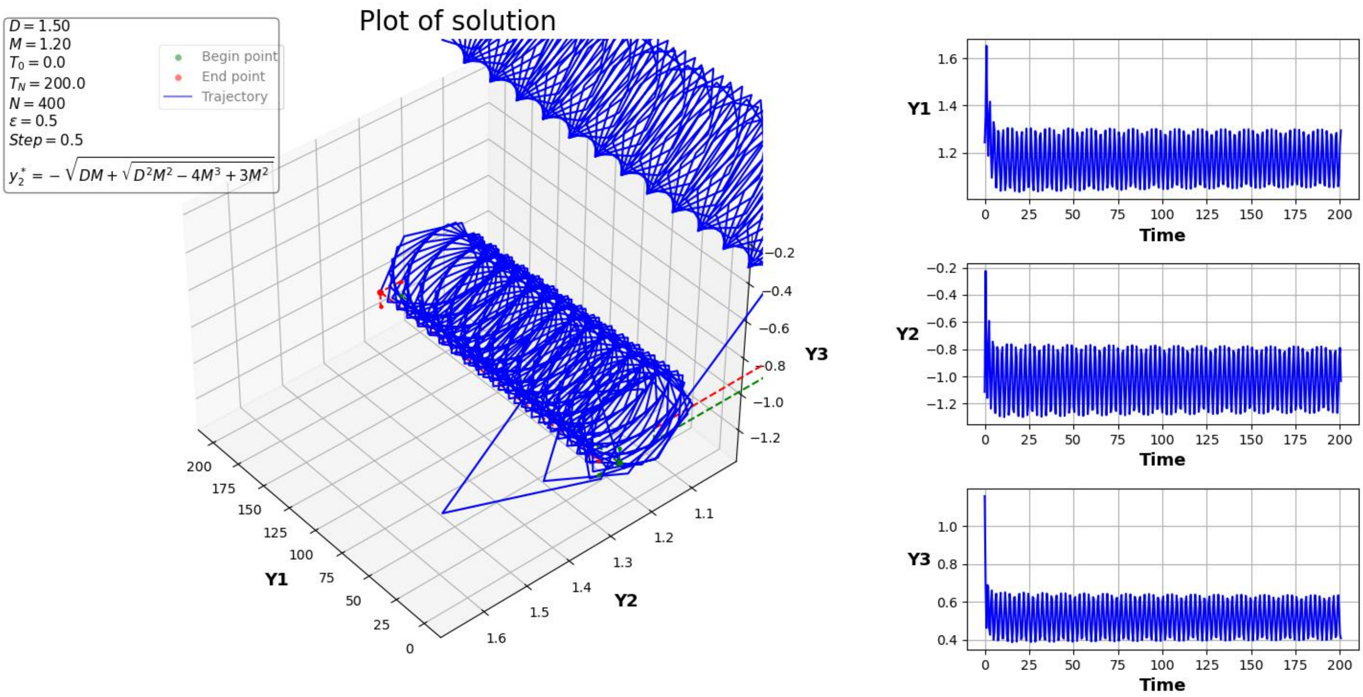

Рис. 1: Периодический режим

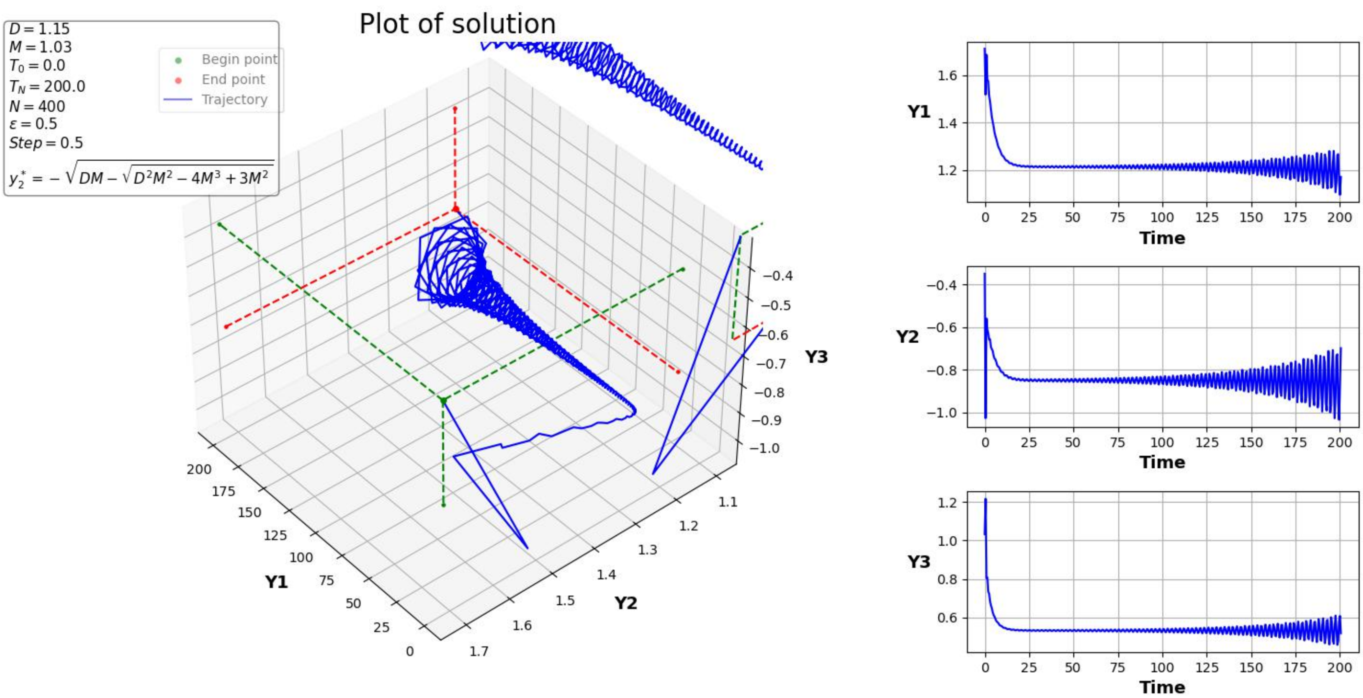

Рис. 2: Колебания с ростом амплитуды

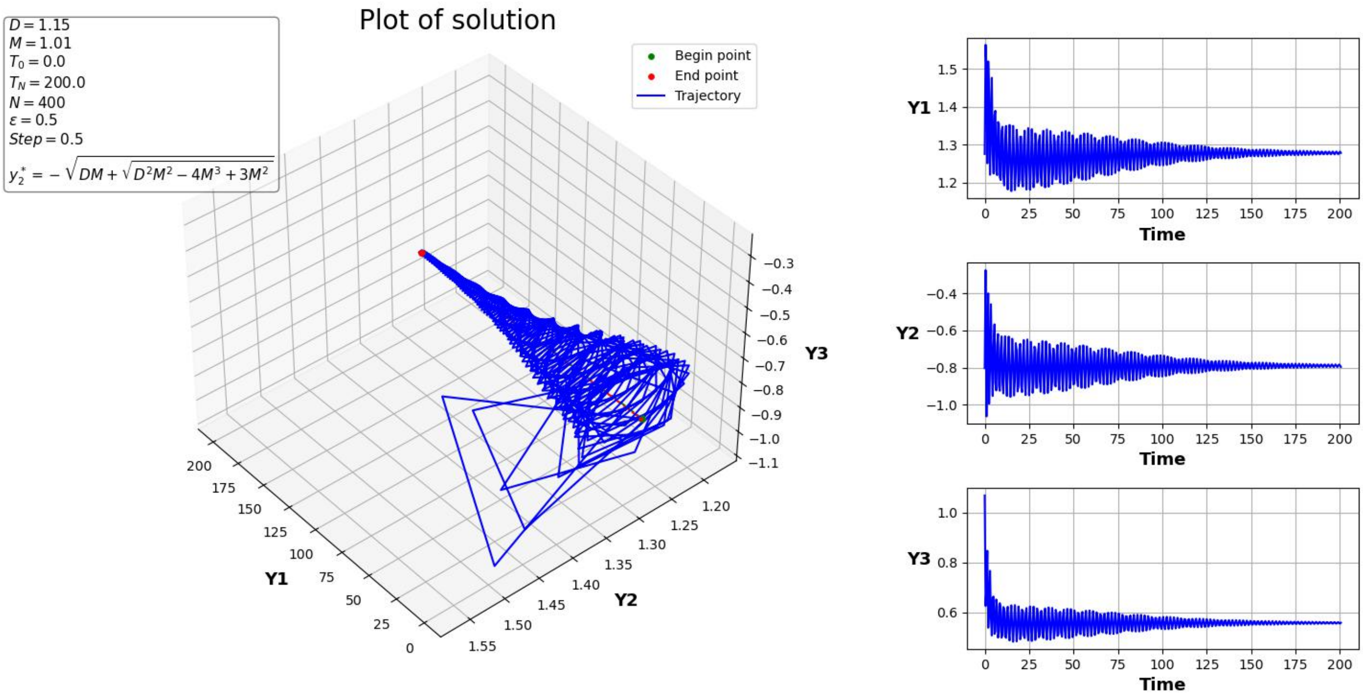

Рис. 3: Колебания с затуханием