

Отчёт по практическому заданию.  
6 семестр

Козлов Кирилл 305 группа

5 мая 2025 г.

# Содержание

# 1 Постановка задачи.

- Требуется решить систему НДС:

$$\begin{cases} \dot{y}_1 = y_1 - \frac{D}{2}y_2 + y_2(y_3 + y_1^2) \\ \dot{y}_2 = \frac{D}{2}y_1 + y_2 + y_1(3y_3 - y_1^2) \\ \dot{y}_3 = -2y_3(M + y_1y_2) \end{cases}$$

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:

$$\begin{cases} 0 = y_1 - \frac{D}{2}y_2 + y_2(y_3 + y_1^2) \\ 0 = \frac{D}{2}y_1 + y_2 + y_1(3y_3 - y_1^2) \\ 0 = -2y_3(M + y_1y_2) \end{cases}$$

Решая систему получим, систему:

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

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

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

# 3 Численное решение задачи

## 3.1 Метод Рунге-Кутты 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:

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

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

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

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

```

1  import numpy as np
2  import pandas as pd
3  import os
4  import matplotlib.gridspec as gridspec
5  import matplotlib.pyplot as plt
6  from ipywidgets import interact, IntSlider, Play, VBox, HBox, FloatSlider
7  import re
8  import math
9  import plotly.graph_objects as go
10 from IPython.display import display, clear_output
11 import ipywidgets as widgets
12
13
14 # удаляем старые данные
15 !rm -rf data_files/data_M*.csv

```

```

16
17 # ==== Константы ====
18 # зафиксировать одну стационарную точку , один шаг eps = 0.5, один шаг
19 # step=0.01 и D >= 2*sqrt(M)
20 T0 = 0.0
21 TN = 200.0
22 TIME_STEP = 0.5
23 EPS = 0.5
24
25 # ==== Класс состояния ====
26 class State:
27     def __init__(self, y1, y2, y3):
28         self.y1 = y1
29         self.y2 = y2
30         self.y3 = y3
31
32     def copy(self):
33         return State(self.y1, self.y2, self.y3)
34
35 # ==== Вычисления начальных условий ====
36 def compute_stationary_solutions(M, D, sign_1, sign_2):
37     y_star = State(0, 0, 0)
38     discriminant = D**2 * M**2 - 4*M**3 + 3*M**2
39
40     try:
41         inner_sqrt = D*M + sign_2*np.sqrt(discriminant)
42
43         if inner_sqrt < 0:
44             print(f"Warning: Подкоренное выражение отрицательное: D={D}, M={M},
45                   inner_sqrt={inner_sqrt}")
46             inner_sqrt = 0 # Чтобы избежать ошибки, подставляем 0
47
48         y_star.y2 = sign_1*np.sqrt(inner_sqrt)
49
50         # Защита от деления на ноль
51         if y_star.y2 == 0:
52             print(f"Warning: Деление на ноль при расчете y1: D={D}, M={M}")
53             y_star.y1 = 0
54         else:
55             y_star.y1 = -M / y_star.y2
56
57         # Аналогично защищаем y3
58         if y_star.y2 == 0:
59             y_star.y3 = 0
60         else:
61             y_star.y3 = D/2 + (M/(y_star.y2**2)) * (1 - M)
62
63     except Exception as e:
64         print(f"Ошибка при расчете стационарных решений: {e}")
65         y_star = State(0, 0, 0) # Если что-то совсем пошло не так, обнуляем
66     return y_star
67
68 def compute_beginning_values(vector_y_star):
69     return State(
70         vector_y_star.y1 + EPS,
71         vector_y_star.y2 + EPS,
72         vector_y_star.y3 + EPS
73     )

```

```

74
75 # ==== Правая часть системы ====
76 def func(t, y, M, D):
77     dy = State(0, 0, 0)
78
79     dy.y1 = y.y1 - (D / 2.0) * y.y2 + y.y2 * (y.y3 + y.y1**2)
80     dy.y2 = (D / 2.0) * y.y1 + y.y2 + y.y1 * (3 * y.y3 - y.y1**2)
81     dy.y3 = -2.0 * y.y3 * (M + y.y1 * y.y2)
82     return dy
83
84 # ==== Метод Рунге-Кутты 4-го порядка ====
85 def runge_kutta_4_system(M, D, sign_1, sign_2):
86     vector_y_star = compute_stationary_solutions(M, D, sign_1, sign_2)
87     vector_y_0 = compute_beginning_values(vector_y_star)
88     t = T0
89     y = vector_y_0
90     results = []
91
92     while (t <= TN + TIME_STEP):
93         results.append((t, y.copy()))
94         if (abs(y.y1) > 1e6 or abs(y.y2) > 1e6 or abs(y.y3) > 1e6):
95             print(f"Warning: Решение ушло в бесконечность на t={t}")
96             break
97
98         k1 = func(t, y, M, D)
99         k2 = func(t + TIME_STEP/2, State(
100             y.y1 + TIME_STEP/2 * k1.y1,
101             y.y2 + TIME_STEP/2 * k1.y2,
102             y.y3 + TIME_STEP/2 * k1.y3
103         ), M, D)
104         k3 = func(t + TIME_STEP/2, State(
105             y.y1 + TIME_STEP/2 * k2.y1,
106             y.y2 + TIME_STEP/2 * k2.y2,
107             y.y3 + TIME_STEP/2 * k2.y3
108         ), M, D)
109         k4 = func(t + TIME_STEP, State(
110             y.y1 + TIME_STEP * k3.y1,
111             y.y2 + TIME_STEP * k3.y2,
112             y.y3 + TIME_STEP * k3.y3
113         ), M, D)
114
115         y.y1 += TIME_STEP/6 * (k1.y1 + 2*k2.y1 + 2*k3.y1 + k4.y1)
116         y.y2 += TIME_STEP/6 * (k1.y2 + 2*k2.y2 + 2*k3.y2 + k4.y2)
117         y.y3 += TIME_STEP/6 * (k1.y3 + 2*k2.y3 + 2*k3.y3 + k4.y3)
118
119         t += TIME_STEP
120
121     return results

```

## 4.2.2 Построение графиков

```

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

```

```

7         'y1': [y.y1 for t, y in results],
8         'y2': [y.y2 for t, y in results],
9         'y3': [y.y3 for t, y in results],
10    }
11
12    df = pd.DataFrame(data)
13
14    fig = plt.figure(figsize=(16, 8))
15    gs = gridspec.GridSpec(3, 2, width_ratios=[2, 1]) # 3 rows ,2 columns
16    ax = fig.add_subplot(gs[:, 0], projection='3d')
17    # Starting and ending points
18    start = df.iloc[0]
19    end = df.iloc[-1]
20
21    # Draw the start and end points
22    ax.scatter([start["Time"]],[start["y1"]], [start["y2"]], [start["y3"]], color="green", s=15,
23    ax.scatter([end["Time"]],[end["y1"]], [end["y2"]], [end["y3"]], color="red", s=15, label="End")
24
25
26    # --- Projections start point ---
27    # for y2-y3
28    ax.plot([start["Time"],end["Time"]],[start["y1"], start["y1"]], [start["y2"], start["y2"]],
29    ax.scatter([end["Time"]],[start["y1"]], [start["y2"]], [start["y3"]], color="green", s=5, zorder=5)
30    # for y1-y2
31    ax.plot([start["Time"],start["Time"]],[start["y1"], start["y1"]], [start["y2"], end["y2"]],
32    ax.scatter([start["Time"]],[start["y1"]], [end["y2"]], [start["y3"]], color="green", s=5, zorder=5)
33    # for y1-y3
34    ax.plot([start["Time"],start["Time"]],[end["y1"], start["y1"]], [start["y2"], start["y2"]],
35    ax.scatter([start["Time"]],[end["y1"]], [start["y2"]], [start["y3"]], color="green", s=5, zorder=5)
36
37    # --- Projections end point ---
38    # for y2-y3
39    ax.plot([end["Time"],start["Time"]],[end["y1"], end["y1"]], [end["y2"], end["y2"]], [start["y3"],
40    ax.scatter([start["Time"]],[end["y1"]], [end["y2"]], [end["y3"]], color="red", s=5, zorder=5)
41    # for y1-y2
42    ax.plot([end["Time"],end["Time"]],[end["y1"], end["y1"]], [end["y2"], start["y2"]], [end["y3"],
43    ax.scatter([end["Time"]],[end["y1"]], [start["y2"]], [end["y3"]], color="red", s=5, zorder=5)
44    # for y1-y3
45    ax.plot([end["Time"],end["Time"]],[start["y1"], end["y1"]], [end["y2"], end["y2"]], [end["y3"],
46    ax.scatter([end["Time"]],[start["y1"]], [end["y2"]], [end["y3"]], color="red", s=5, zorder=5)
47
48
49    # Make trajectory for y1, y2, y3
50    ax.plot(df["Time"],df["y1"], df["y2"], df["y3"], label="Trajectory", color="blue",linewidth=2)
51
52    ax.set_xlabel("Y1", labelpad=10, fontsize=13,fontweight='bold')
53    ax.set_ylabel("Y2", labelpad=10, fontsize=13,fontweight='bold')
54    ax.set_zlabel("Y3", labelpad=10, fontsize=13,fontweight='bold')
55    ax.set_title("График решения", fontsize=20)
56
57    # Add legend and grid
58    ax.legend(loc='best')
59    ax.view_init(elev=40, azimuth=140)
60    ax.grid(True)
61
62    handles, labels = ax.get_legend_handles_labels()
63    unique_labels = dict(zip(labels, handles))
64    ax.legend(unique_labels.values(), unique_labels.keys())

```

```

65
66
67 ax2 = fig.add_subplot(gs[0, 1])
68 ax2.plot(df["Time"], df["y1"], color='blue')
69 ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
70 ax2.set_ylabel("y1".upper(), rotation=0, labelpad=13, fontsize=13,fontweight='bold')
71 ax2.grid(True)
72
73 ax2 = fig.add_subplot(gs[1, 1])
74 ax2.plot(df["Time"], df["y2"], color='blue')
75 ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
76 ax2.set_ylabel("y2".upper(), rotation=0, labelpad=13,fontsize=13,fontweight='bold')
77 ax2.grid(True)
78
79 ax2 = fig.add_subplot(gs[2, 1])
80 ax2.plot(df["Time"], df["y3"], color='blue')
81 ax2.set_xlabel("Time",fontsize=13,fontweight='bold')
82 ax2.set_ylabel("y3".upper(), rotation=0, labelpad=13,fontsize=13,fontweight='bold')
83 ax2.grid(True)
84
85 # Add text with constants to the left upper corner
86 formula_str = (
87     f'$y_2^* = {" " if sign_1 == 1 else "-"}\sqrt{{DM {"+ " if sign_2 == 1 else "-"} }\sqrt{{D
88 )
89 textstr = '\n'.join((
90     r'$D= %s$' % D,
91     r'$M= %s$' % M,
92     r'$T_0= %.1f$' % T0,
93     r'$T_N= %.1f$' % TN,
94     r'$\varepsilon= %.1f$' % EPS,
95     r'$STEP= %.1f$' % TIME_STEP,
96     formula_str
97 ))
98 props = dict(boxstyle='round', facecolor='white', alpha=0.5)
99
100 # Настройка отступов
101 plt.subplots_adjust(left=0.05, right=0.95, top=0.95, bottom=0.1, hspace=0.4)
102 plt.gcf().text(0.01, 0.8, textstr, fontsize=13, bbox=props)
103
104 plt.show()
105 else:
106     print(f"Некорректные 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  $D > 2.0, M > 2.0$ .



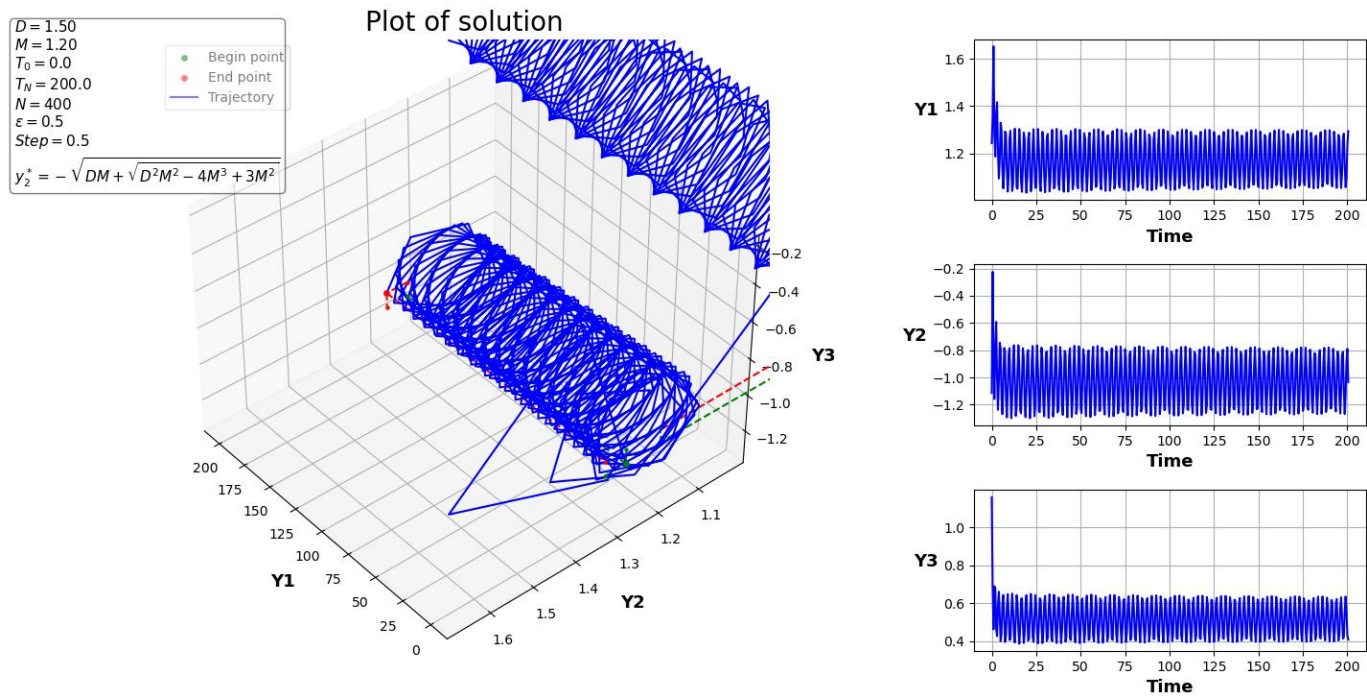


Рис. 1: Периодический режим

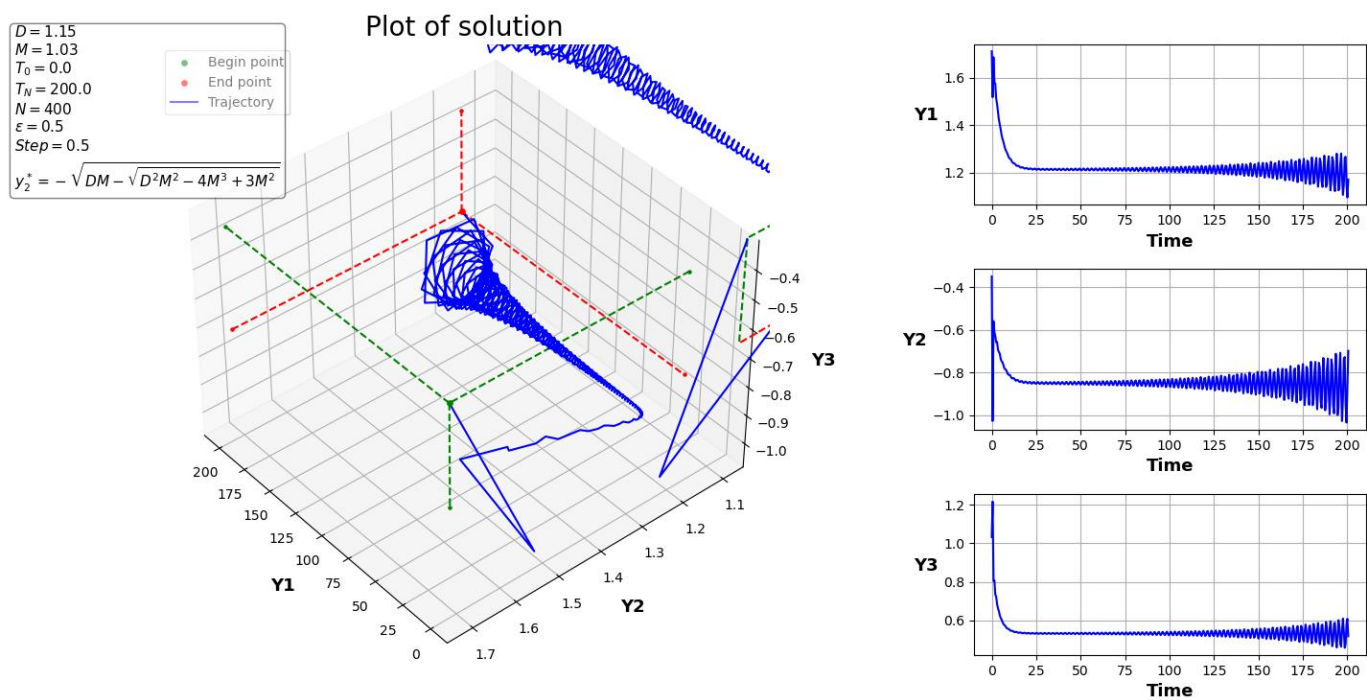


Рис. 2: Колебания с ростом амплитуды

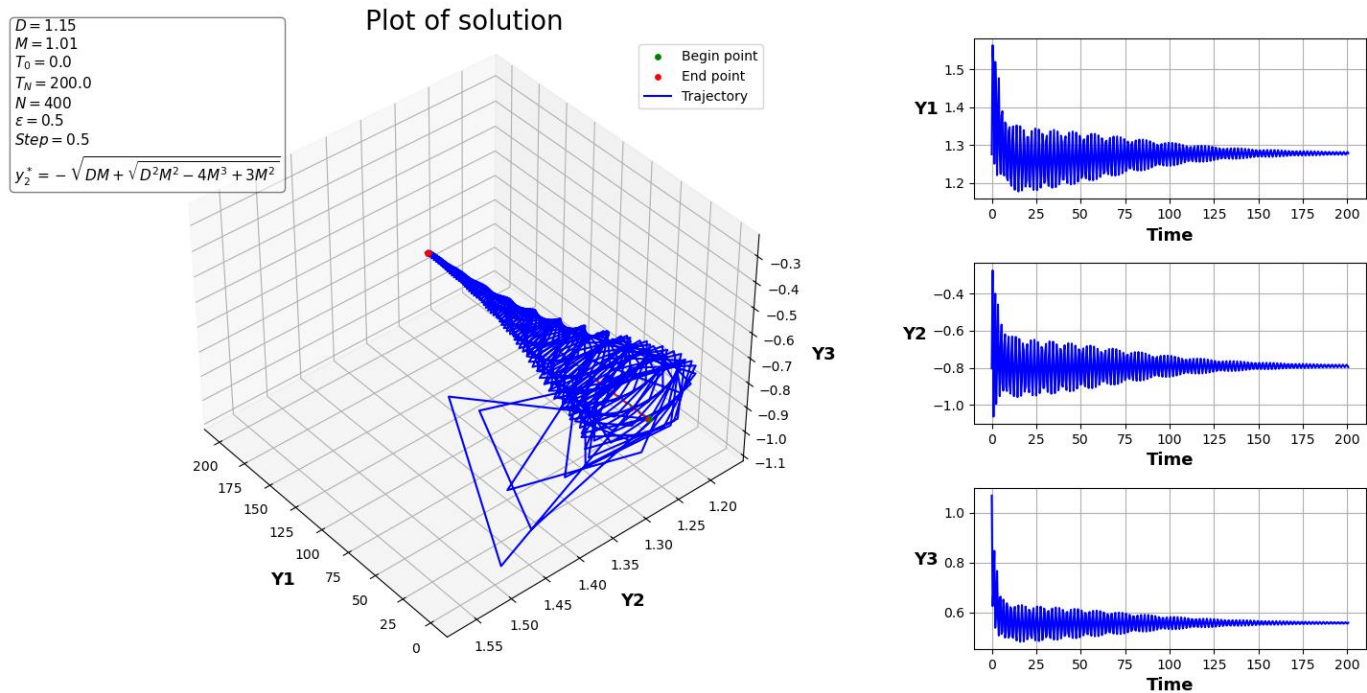


Рис. 3: Колебания с затуханием