**Отчет по курсовой работе  
по дисциплине «Математическое моделирование»  
Вариант №1**

Выполнил студент  
гр. 5130904/30007 Голиков В.С.

Руководитель Устинов С.М.

**Оглавление**

[1. Формулировка задачи 3](#_Toc197945702)

[2. Блок аналитических преобразований 4](#_Toc197945703)

[3. Текст программы 5](#_Toc197945704)

[4. Результаты (таблицы и графики) 8](#_Toc197945705)

[5. Блок проверки 11](#_Toc197945706)

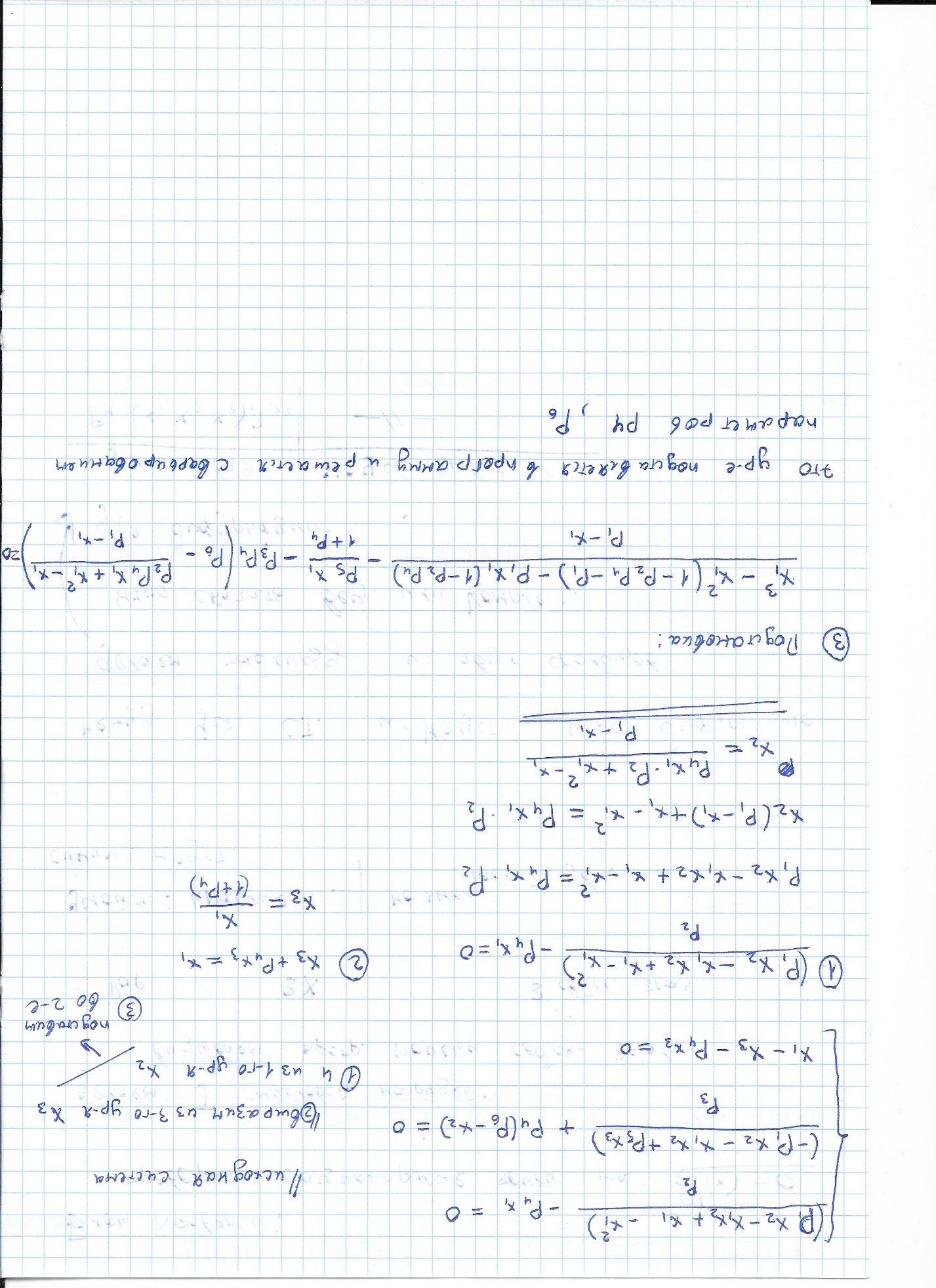
[6. Вывод 12](#_Toc197945707)

# 1. Формулировка задачи

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Контент, сгенерированный ИИ, может содержать ошибки.

# 2. Блок аналитических преобразований



# 3. Текст программы (среда PyCharm, язык Python)

from sympy import symbols, Eq, solve  
import matplotlib.pyplot as plt  
import numpy as np  
import pandas as pd  
  
# параметры  
p1 = 8.4E-6  
p2 = 6.6667E-4  
p3 = 1.7778E-5  
p5 = 2  
  
def check(x1, x2, x3, p4, p6):  
 d1 = ((p1\*x2 - x1\*x2 + x1 - x1\*\*2)/p2) - p4\*x1  
 d2 = ((-p1\*x2 - x1\*x2 + p5\*x3)/p3) + p4\*(p6-x2)  
 d3 = x1 - x3 - p4\*x3  
 return (d1 <=10\*\*-13 and d2 <=10\*\*-13 and d3 <=10\*\*-13)  
  
def jacobieigenval(j\_x1, j\_x2, j\_p4):  
 a11 = ((-j\_x2 + 1 - 2 \* j\_x1) / p2) - j\_p4  
 a12 = (p1 - j\_x1) / p2  
 a13 = 0.0  
 a21 = -j\_x2 / p3  
 a22 = ((-p1 - j\_x1) / p3) - j\_p4  
 a23 = p5 / p3  
 a31 = 1.0  
 a32 = 0.0  
 a33 = -1 - j\_p4  
  
 matrix = np.array([  
 [float(a11), float(a12), float(a13)],  
 [float(a21), float(a22), float(a23)],  
 [float(a31), float(a32), float(a33)]  
 ])  
  
 eigenval = np.linalg.eigvals(matrix)  
 return eigenval  
  
x1 = symbols('x1')  
  
# Список диапазонов p4  
p4\_ranges = [0.01, 0.1, 1, 10, 100]  
  
for p6 in [10, 100]:  
 all\_P4 = []  
 all\_X1 = []  
 all\_X2 = []  
 all\_X3 = []  
 all\_eigenval = []  
  
 for delim in p4\_ranges:  
 for i in range(1, 11):  
 p4 = i \* delim  
  
 equation = Eq(((x1 \*\* 3) - (x1 \*\* 2) \* (1 - p2 \* p4 - p1) - p1 \* x1 \* (1 - p2 \* p4)) / (p1 - x1) - (  
 (p5 \* x1) / (1 + p4)) - p3 \* p4 \* (p6 - (p2 \* p4 \* x1 + (x1 \*\* 2) - x1) / (p1 - x1)), 0)  
  
 solutionX1 = solve(equation, x1)  
 sol = [complex(x).real for x in solutionX1]  
 solutionX1 = []  
 for x1\_value in sol:  
 if x1\_value > 0:  
 solutionX1.append(x1\_value)  
  
 solutionX2 = []  
 solutionX3 = []  
 current\_eigenvals = []  
  
 for x1\_val in solutionX1:  
 x2 = (p4 \* x1\_val \* p2 + x1\_val \*\* 2 - x1\_val) / (p1 - x1\_val)  
 x3 = x1\_val / (1 + p4)  
  
 solutionX2.append(x2)  
 solutionX3.append(x3)  
 eigenvalues = jacobieigenval(x1\_val, x2, p4)  
 current\_eigenvals.append(eigenvalues)  
  
 all\_P4.append(p4)  
 all\_X1.append(solutionX1)  
 all\_X2.append(solutionX2)  
 all\_X3.append(solutionX3)  
 all\_eigenval.append(current\_eigenvals)  
  
  
 # Функция для проверки устойчивости точки  
 def is\_stable(eigenvalues):  
 return all(np.real(eig) < 0 for eig in eigenvalues)  
  
 #построение графиков  
 def plot\_log\_graph(x\_data, y\_data, eigenvals, title, ylabel):  
 plt.figure(figsize=(10, 6))  
  
 for p4, y\_vals, evals in zip(x\_data, y\_data, eigenvals):  
 colors = ['green' if is\_stable(e) else 'red' for e in evals]  
 plt.scatter([p4] \* len(y\_vals), y\_vals, c=colors, marker='o', s=20)  
  
 plt.xlabel('P4', fontweight='bold', fontsize=12)  
 plt.ylabel(ylabel, fontweight='bold', fontsize=12)  
 plt.title(title, pad=20, fontsize=14)  
 plt.xscale('log')  
 plt.yscale('log')  
 plt.grid(True, linestyle='--', alpha=0.5, which='both')  
 plt.tight\_layout()  
 plt.show()  
  
 plot\_log\_graph(all\_P4, all\_X1, all\_eigenval, f'X1(P4) при P6 = {p6}', 'X1 (log scale)')  
 plot\_log\_graph(all\_P4, all\_X2, all\_eigenval, f'X2(P4) при P6 = {p6}', 'X2 (log scale)')  
 plot\_log\_graph(all\_P4, all\_X3, all\_eigenval, f'X3(P4) при P6 = {p6}', 'X3 (log scale)')

# Создание таблицы  
 data = {  
 'p4': [],  
 'x1': [],  
 'x2': [],  
 'x3': [],  
 'l1': [],  
 'l2': [],  
 'l3': [],  
 }  
  
 for p4, x1\_vals, x2\_vals, x3\_vals, eigenvals in zip(all\_P4, all\_X1, all\_X2, all\_X3, all\_eigenval):  
 for i in range(len(x1\_vals)):  
 data['p4'].append(p4)  
 data['x1'].append(x1\_vals[i])  
 data['x2'].append(x2\_vals[i])  
 data['x3'].append(x3\_vals[i])  
 data['l1'].append(eigenvals[i][0])  
 data['l2'].append(eigenvals[i][1])  
 data['l3'].append(eigenvals[i][2])  
  
 df = pd.DataFrame(data)  
 filename = f"p4\_full\_range\_p6\_{p6}\_table.xlsx"  
 df.to\_excel(filename, index=False, engine='openpyxl')

# 4. Результаты (таблицы и графики)



P.S. Это встроенный в Word Excel файл, двойным нажатием на иконку можно просмотреть таблицу значений, полученных в результате выполнения программы. Файл содержит два листа:   
1. Значения x1,x2,x2 и собственные значения матрицы Якоби для каждой точки при P6 = 10  
2. Значения x1,x2,x2 и собственные значения матрицы Якоби для каждой точки при P6 = 100

**Графики (красным – неустойчивые, зеленым - устойчивые)**

**Изображение выглядит как текст, График, линия, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, диаграмма, линия, График

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, График, линия, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.**

**Изображение выглядит как текст, линия, График, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.**

# 5. Блок проверки

Для проверки полученных значений самостоятельно реализовал функцию check(x1, x2, x3, p4, p6)  
Проверять будем исходные функции, а точнее их производные из системы. Если при подстановке мы получим 0 или близкое к 0 число (из-за погрешности вычислений), то точка действительно является стационарной

Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

При вызове этой функции внутри цикла получения значений, мы ожидаем увидеть результат «True», что будет означать совпадение с 0, а значит и точка является стационарной.

Ниже привожу скриншот выполнения этой функции и ее вызов

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

# 6. Вывод

В результате проделанной работы мною была написана программа на языке программирования Python, которая по заданной системе вычисляет стационарные точки в зависимости от параметров P1-P6. Далее я построил матрицу Якоби и изучил ее собственные значения для оценки устойчивости полученных стационарных точек. Результатом работы программы являются 6 графиков зависимости каждого X от P4 и P6, а так же таблица значений, состоящая из двух листов для разделения значений, зависящих от параметра P6.

По таблице я нашел промежутки бифуркации и оценил ее вид. Для примера, привожу скриншоты разных видов бифуркации (на скриншотах первые столбец – значение параметра P4, далее 3 столбца – x1, x2, x3 соответственное, далее – собственные значения матрицы Якоби):

**Вещественная**: (одно из собственных значений переходит через 0)



**Комплексная**: (вещественная часть переходит через 0)



Таким образом, можно наблюдать переход состояния точки из устойчивого в неустойчивое. На графике эта точка не отмечается, так как явно ее не существует в таблице, однако при выборе маленького шага можно найти точку бифуркации (место, где и происходит переход, то есть собственное значение/его вещественная часть становится равным 0)

По окончании работы так же была написана функция check(x1, x2, x3, p4, p6) для проверки стационарных точек (производные из системы равны 0 при подстановке значений). Выполнив эту функцию, можно заметить, что все полученные значения удовлетворяют условию.

В заключении, поставленная задача выполнена полностью с учетом замечаний