Лабораторная работа №4. Явные методы Рунге-Кутты

Денис Слынко, Б01-818

2021

Задача:

$$\begin{cases} u' = A + u^2 v - (B+1)v, u(0) = 1; \\ v' = Bu - u^2 v, v(0) = 1, \\ A = 1, B \in [1, 5]. \end{cases}$$

Получить численное решение системы явными методами Рунге–Кутты первого и четвертого порядков. Изучить фазовые портреты. Удалось ли Вам получить предельные циклы и бифуркацию Хопфа (при которой предельный цикл вырождается в точку; при этом $B \to A(A+1)$)? Эта система – модель Лефевра–Пригожина «брюсселятор».

1 Используемые методы Рунге-Кутты

В условии требуется решить задачу явными методами первого и четвертого порядков сходимости. В качестве первого метода был выбран метод со следующий таблицей Бутчера:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & \\ \hline & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{array}$$

Как видно, такой метод удовлетворяет условию Кутты

$$c_j = \sum_{k=1}^{s} a_{jk}, j = 1, ..., s \tag{1}$$

и условию первого порядка сходимости

$$\sum_{k=1}^{s} b_k = 1. \tag{2}$$

При этом условие второго порядка уже не выполняется:

$$\sum_{j=1}^{s}\sum_{k=1}^{s} b_j a_{jk} = \frac{3}{8} \neq \frac{1}{2}. \tag{3}$$

В качестве второго метода (с четвертым порядком сходимости) был выбран метод со следующий таблицей Бутчера:

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

2 Результаты вычислений

Вычисления численного решения системы ОДУ проводились в промежутке $[0, 10]$ с шагом 0.02 и 0.001. Значения B выбирались в промежутке $[1, 5]$ с шагом 0.5. Были построены графики решений, полученных обоими методами, в зависимости от параметра x и фазовые траектории соответствующих решений. Ниже приведены некоторые из полученных графиков.

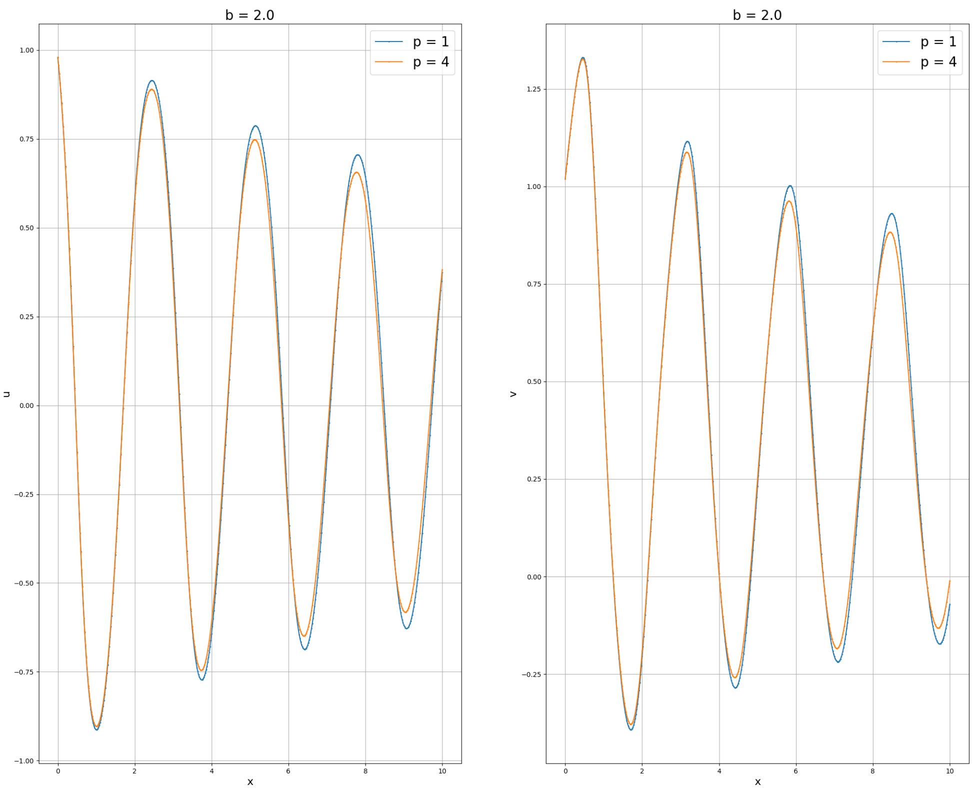

Рис. 1: Решения для $h = 0.02, B = 2.0$

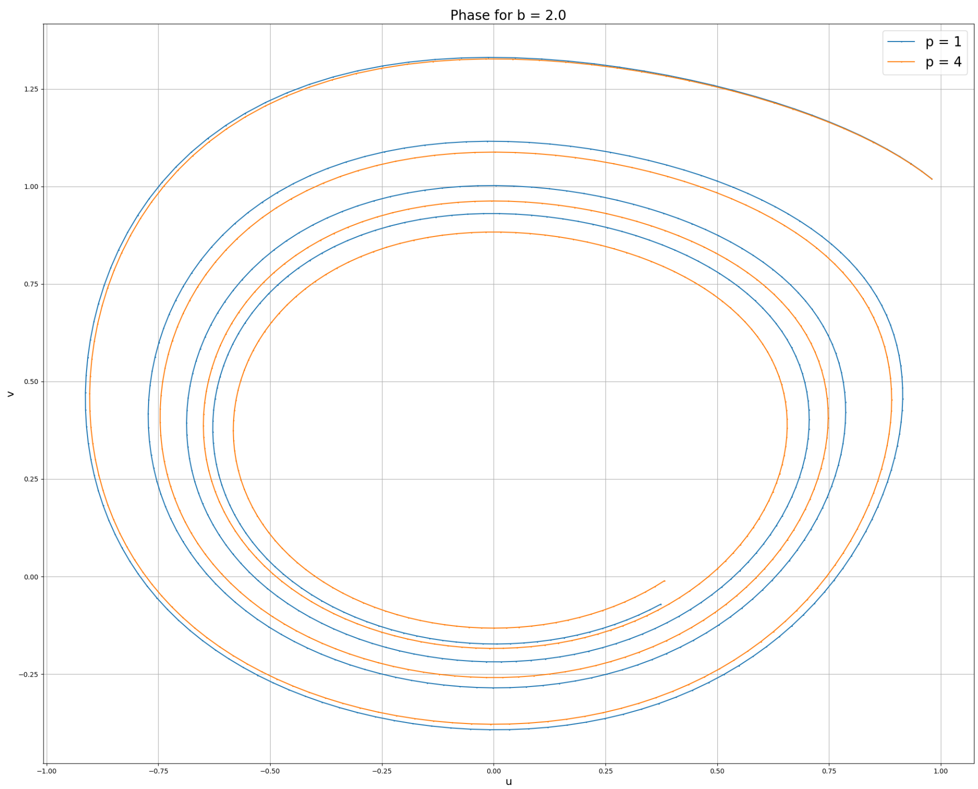

Рис. 2: Фазовые траектории для $h = 0.02, B = 2.0$

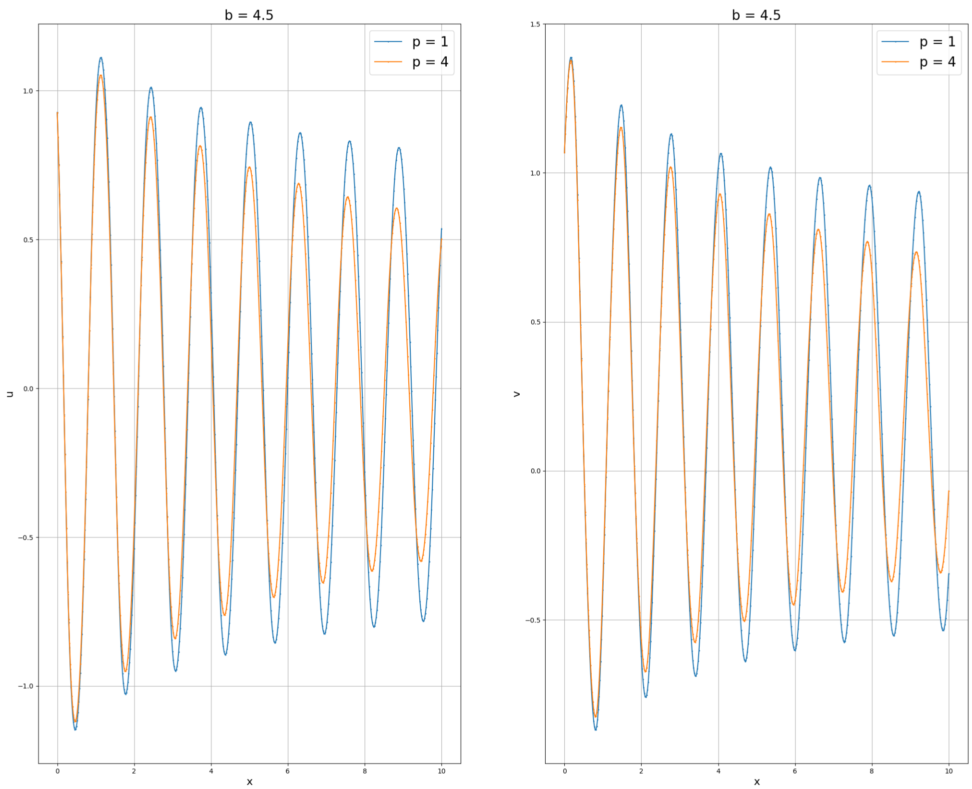

Рис. 3: Решения для $h = 0.02, B = 4.5$

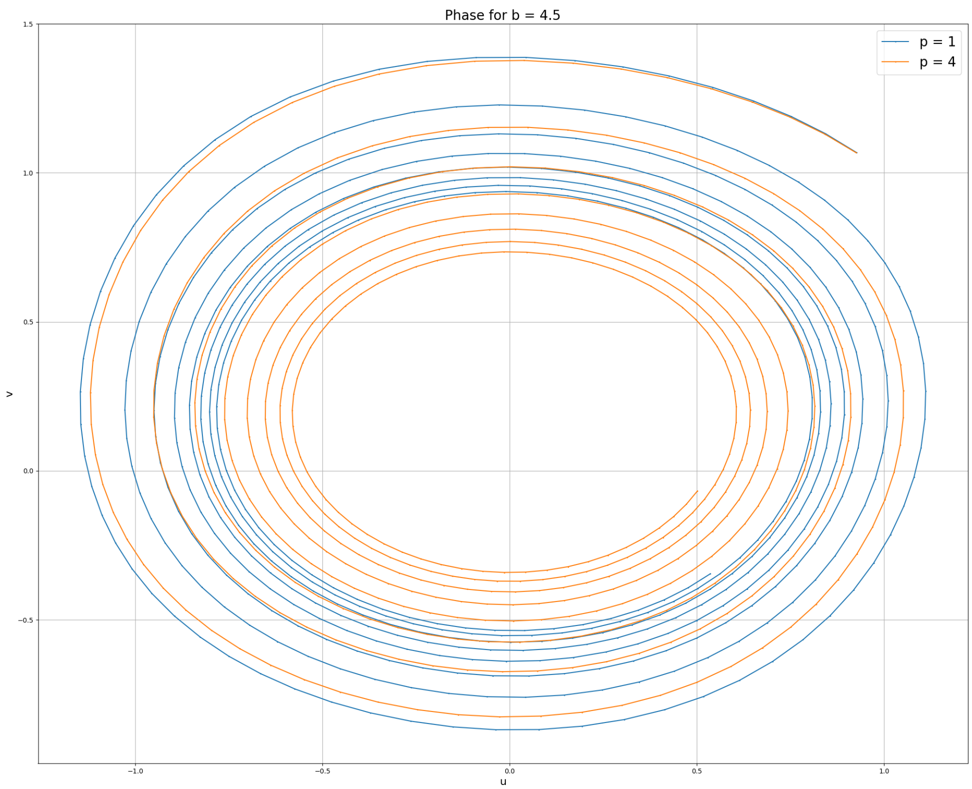

Рис. 4: Фазовые траектории для $h = 0.02, B = 4.5$

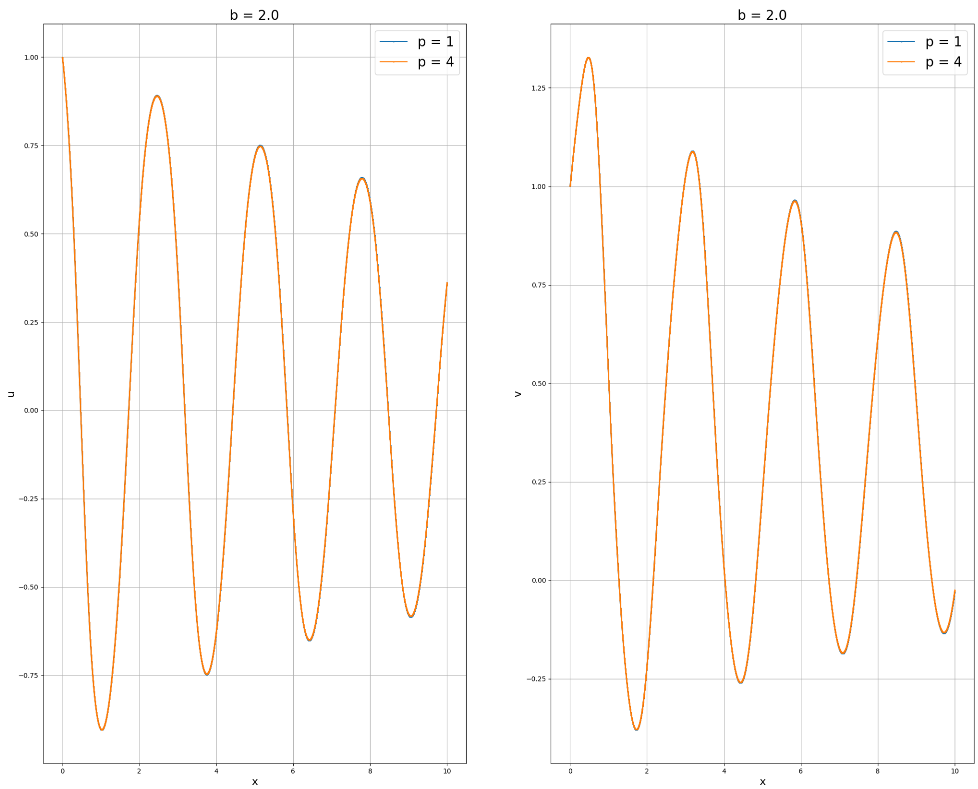

Рис. 5: Решения для $h = 0.001, B = 2.0$

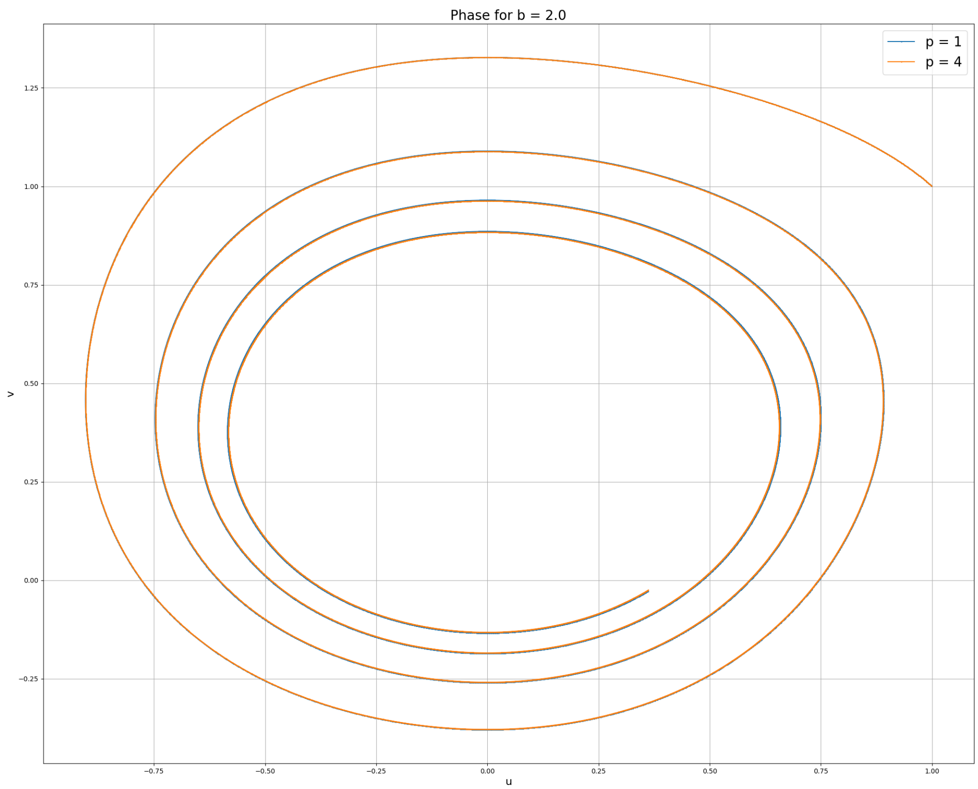

Рис. 6: Фазовые траектории для $h = 0.001, B = 2.0$

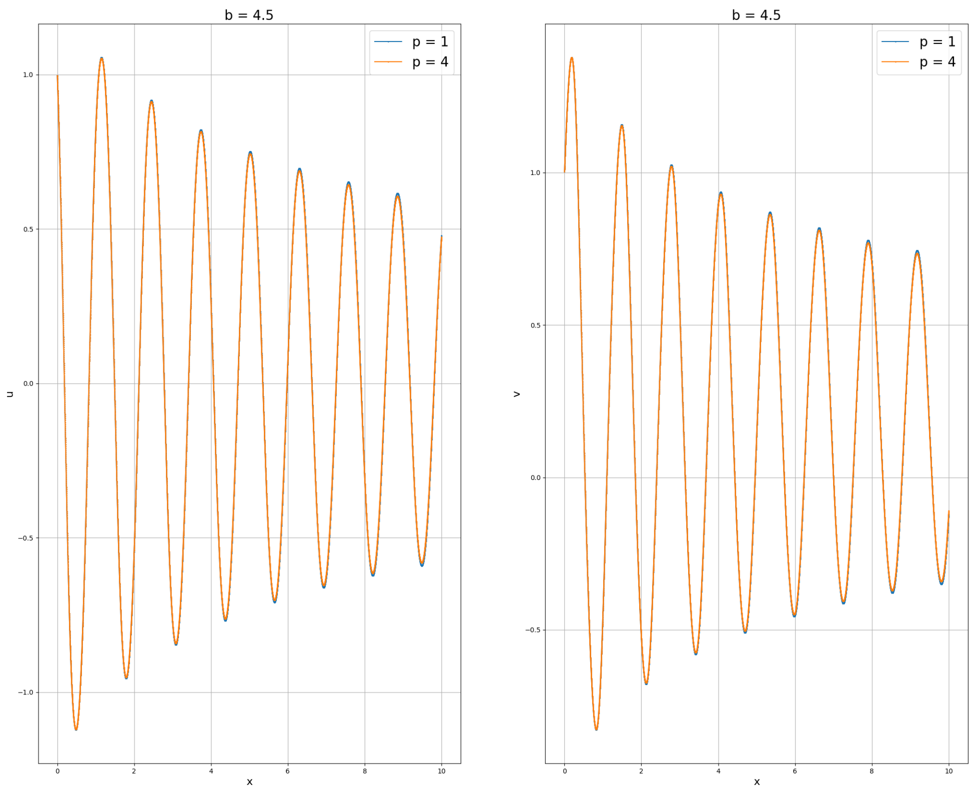

Рис. 7: Решения для $h = 0.001, B = 4.5$

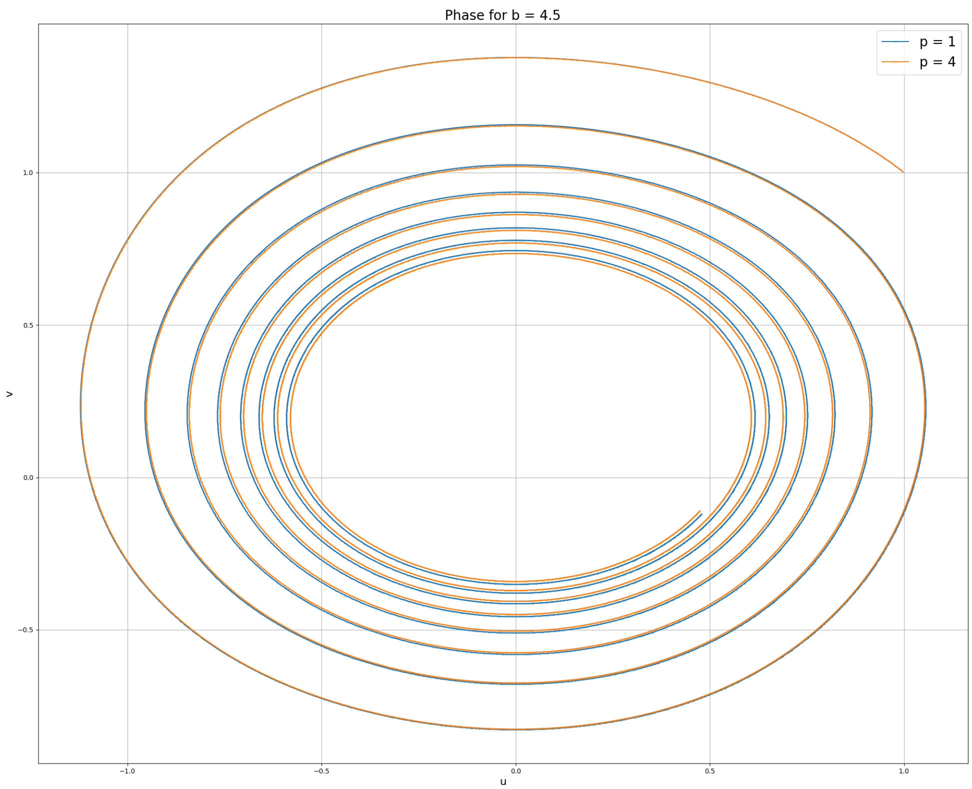

Рис. 8: Фазовые траектории для $h = 0.001, B = 4.5$

Как видно из графиков, при увеличении параметра B скорость скручивания увеличивается. В пределе $x \to +\infty$ получаем предельный цикл (бифуркация Хопфа):

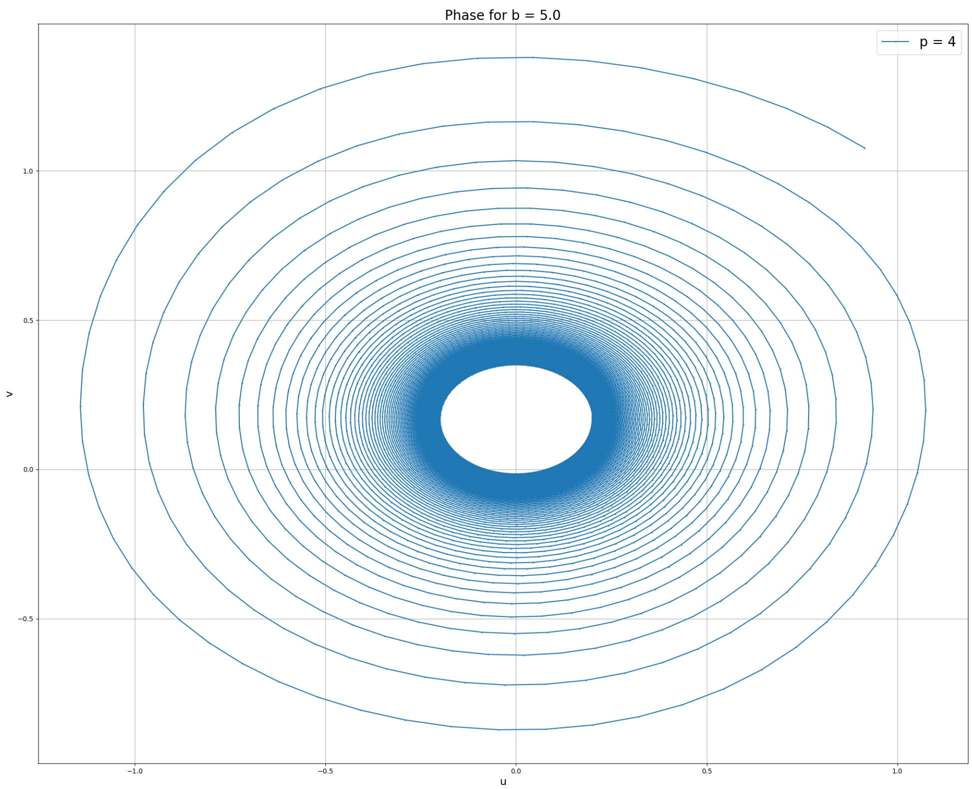

Рис. 9: Фазовые траектории для $h = 0.001, B = 5.0$ в пределе $x \to +\infty$

3 Код программы

// Main.py

```
import numpy as np
import matplotlib.pyplot as plt
import subprocess
import equation
import sys
```

```python
import os


class Main:
    system = None
    num_of_steps = 0
    right_end = 0

    def __init__(self, right_end=1.0):
        self.system = equation.Equation()
        self.num_of_steps = 4
        self.right_end = right_end

    @staticmethod
    def create_plots_dir(name="Plots"):
        if not
            os.path.isdir(os.path.join(os.path.dirname(sys.argv[0]),
            name)):
            subprocess.check_output(f"mkdir {name}", shell=True)

    def __sub_solve__(self, u_n, h, b, p):
        if p == 1:
            alpha = np.zeros(self.num_of_steps *
                self.num_of_steps).reshape(self.num_of_steps,
                self.num_of_steps)
            alpha[1][0] = alpha[2][1] = alpha[3][2] = 0.5
            # deprecated
            c = np.array([0, 0.5, 0.5, 0.5])
            beta = np.array([0.25, 0.25, 0.25, 0.25])
        elif p == 4:
            alpha = np.zeros(self.num_of_steps *
                self.num_of_steps).reshape(self.num_of_steps,
                self.num_of_steps)
            alpha[1][0] = alpha[2][1] = 0.5
            alpha[3][2] = 1.0
            # deprecated
            c = np.array([0, 0.5, 0.5, 1])
            beta = np.array([1.0 / 6.0, 1.0 / 3.0, 1.0 / 3.0, 1.0 /
                6.0])
            # Another possible method - '3/8 rule'
```

```
        # beta = np.array([1.0 / 8.0, 3.0 / 8.0, 3.0 / 8.0, 1.0 /
            8.0])
        # alpha[1][0] = 1.0 / 3.0
        # alpha[2][0] = -1.0 / 3.0
        # alpha[2][1] = alpha[3][0] = alpha[3][2] = 1.0
        # alpha[3][1] = -1.0
    else:
        raise ValueError(f"Unsupported order: p = {p}")

    k = np.zeros(self.num_of_steps *
        self.system.num_of_equations).reshape(self.num_of_steps,
                                                                                        self.system.num_of
    for i in range(self.num_of_steps):
        tmp = alpha[i][0] * k[0]
        if i > 0:
            for j in range(1, i):
                tmp += alpha[i][j] * k[j]
        k[i] = self.system.calculate_f(u_n + h * tmp, b)

    # print(f"Step for {x_n}: {u_n + h * c * sum(x for x in k)}")
    return u_n + h * beta.dot(k)

def solve(self, h, b, p=4):
    x_axis = []
    u_values = ([], [])

    x = 0.0
    u = self.system.initial_conditions
    while x <= self.right_end:
        x_axis.append(x)
        u = self.__sub_solve__(u, h, b, p)
        x += h
        for i in range(self.system.num_of_equations):
            u_values[i].append(u[i])

    return x_axis, u_values

def main(self, b, h=0.02, name="Plots"):
    Main.create_plots_dir(name=name)
```

```
fig = plt.figure()

x_axis, u_values = self.solve(h, b, p=1)
x_axis4, u_values4 = self.solve(h, b, p=4)

ax = fig.add_subplot(121)
ax.plot(x_axis, u_values[0], label='p = 1', marker='o',
    markersize=1)
ax.plot(x_axis4, u_values4[0], label='p = 4', marker='o',
    markersize=1)
ax.set_title(f"b = {b}", fontsize=20)
ax.set_xlabel("x", fontsize=16)
ax.set_ylabel("u", fontsize=16)
ax.legend(fontsize=20)
ax.grid()

ax = fig.add_subplot(122)
ax.plot(x_axis, u_values[1], label='p = 1', marker='o',
    markersize=1)
ax.plot(x_axis4, u_values4[1], label='p = 4', marker='o',
    markersize=1)
ax.set_title(f"b = {b}", fontsize=20)
ax.set_xlabel("x", fontsize=16)
ax.set_ylabel("v", fontsize=16)
ax.legend(fontsize=20)
ax.grid()

fig.set_figheight(20)
fig.set_figwidth(25)
fig.savefig(os.path.join(name, f"{str(b).replace('.',
    '_')}.png"))

fig = plt.figure()

ax = fig.add_subplot(111)
ax.plot(u_values[0], u_values[1], label='p = 1', marker='o',
    markersize=1)
ax.plot(u_values4[0], u_values4[1], label='p = 4',
    marker='o', markersize=1)
ax.set_title(f"Phase for b = {b}", fontsize=20)
```

```
        ax.set_xlabel("u", fontsize=16)
        ax.set_ylabel("v", fontsize=16)
        ax.legend(fontsize=20)
        ax.grid()

        fig.set_figheight(20)
        fig.set_figwidth(25)
        fig.savefig(os.path.join(name, f"Phase_{str(b).replace('.',
            '_')}.png"))


if __name__ == '__main__':
    main = Main(10.0)
    b = 1.0
    while b <= 5.0:
        main.main(b, h=0.001, name="new")
        b += 0.5
```

// equation.py

```
import numpy as np


class Equation:
    num_of_equations = 0
    equations = []
    initial_conditions = None

    def __init__(self):
        self.num_of_equations = 2

        self.equations.append("x1' = 1 + x1^2 * v - (B - 1) * x2")
        self.equations.append("x2' = B * x1 - x1^2 * x2")

        self.initial_conditions = np.array([1.0, 1.0])

    def get_equations(self):
        for eq in self.equations:
            print(eq)
```

```
        for i in range(len(self.initial_conditions)):
            print(f"x{i + 1}({self.initial_conditions[i][0]}) =
                {self.initial_conditions[i][1]}")

    def calculate_f(self, u, b):
        if b < 1.0 or b > 5.0:
            raise ValueError(f"b = {b} doesn't match the task range
                [1, 5]")

        return np.array([1.0 + u[0] * u[0] * u[1] - (b + 1) * u[1],
            b * u[0] - u[0] * u[0] * u[1]])
```

4 Литература

1 Практические занятия по вычислительной математике : учебное пособие / Е.Н. Аристова, Н.А. Завьялова, А.И. Лобанов. Часть I. – М. : МФТИ, 2014. – 243 с. ISBN 978-5-7417-0541-4 (Ч. 1)