Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

К У Р С О В А Я   Р А Б О Т А

по дисциплине "Математические модели"
вариант 10

Выполнил
студент гр. 3530904/80001: Лукина В. А.

Руководитель
профессор, д.т.н. Устинов С. М.

«___»____________ 2020 г.

Санкт-Петербург
2020

1 Постановка задачи

Для `МОДЕЛИ 8` в плоскости параметров (p_3, p_2) построить бифуркационные диаграммы точек вещественной бифуркации для $p_1 = 2, p_4 = 10 * p_3$. Значения $p2, p3 > 0$. Может оказаться более наглядным использовать логарифмический масштаб по оси p_3. Проиллюстрировать количество решений в каждой области.

$$
\begin{aligned}
\frac{dx_1}{dt} &= p_1 - (p_2 + 1)x_1 + x_1^2 x_2 + p_3(x_3 - x_1);\\
\frac{dx_2}{dt} &= p_2 x_1 - x_1^2 x_2 + p_4(x_4 - x_2);\\
\frac{dx_3}{dt} &= p_1 - (p_2 + 1)x_3 + x_3^2 x_4 - p_3(x_3 - x_1);\\
\frac{dx_4}{dt} &= p_2 x_3 - x_3^2 x_4 - p_4(x_4 - x_2);
\end{aligned}
$$

2 Блок аналитических преобразований

Построим матрицу Якоби системы:

$$\begin{pmatrix} -p_2-1+2x_1x_2-p_3 & x_1^2 & p_3 & 0 \\ p_2-2x_1x_2 & -x_1^2-p_4 & 0 & p_4 \\ p_3 & 0 & -p_2-1+2x_3x_4-p_3 & x_3^2 \\ 0 & p_4 & p_2-2x_3x_4 & -x_3^2-p_4 \end{pmatrix}$$

Так как точки вещественной бифуркации являются стационарными точками, то производная в них равна нулю. Поэтому приравняем правые части исходных уравнений к нулю. Кроме того условием вещестенной бифуркации является то, что одно собственное значение матрицы Якоби становится равным нулю, следовательно и сам определитель равен нулю. Это и будет пятым уравнением. Получаем 5 уравнений и 6 неизвестных. Возьмём x_1 в качестве варируемой переменной и выразим через неё остальные неизвестные.
Для того, чтобы выразить x_3 через x_1 сложим все 4 исходных уравнения и получим уравнение (*):

$$x_3 = 4 - x_1 \tag{*}$$

Для того, чтобы выразить x_4 через x_1 сложим 1 и 2 уравнение, 3 и 4 уравнение, а результаты вычтем и получим уравнение (**):

$$x_4 = x_2 + \frac{(1+2p_3)(x_1-2)}{10p_3} \tag{**}$$

Выразим x_2 из 2-го уравнения и получим уравнение (***):

$$x_2 = \frac{(1+2p_3)(x_1-2)+p_2x_1}{x_1^2} \tag{***}$$

Теперь (*),(**),(***) подставляем в 4-ое уравнение и получаем уравнение (****), которое является линейным относительно p_2 и квадратным относительно p_3:

$$(x_1-2)((20x_1^2+20x_3^2)p_3^2+(10x_1^2+10x_3^2+2x_1^2x_3^2-20x_1x_3p_2)p_3+x_1^2x_3^2)=0 \tag{****}$$

Теперь, если подставим уравнения (*),(**),(***) мы также получим уравнение относительно p_2 и p_3. Тогда определитель и (****) будут решаться совместно. Для этого сначала упростим наш определитель:
Заменим 1 строку на сумму 4-х, 3 строку на сумму 3 и 4:

$$\begin{vmatrix} -1 & 0 & -1 & 0 \\ p_2-2x_1x_2 & -x_1^2-p_4 & 0 & p_4 \\ p_3 & p_4 & -1-p_3 & -p_4 \\ 0 & p_4 & p_2-2x_3x_4 & -x_3^2-p_4 \end{vmatrix}$$

Затем вычтем из 3-го столбца 1-ый, а ко 2-ому добавим 4-ый и разложим определитель по 1-ой строке:

$$\begin{vmatrix} -x_1^2 & -p_2+2x_1x_2 & 10p_3 \\ 0 & -1-2p_3 & -10p_3 \\ -x_3^2 & p_2-2x_3x_4 & -x_3^2-10p_3 \end{vmatrix}$$

Считаем определитель и подставляем (*), (**), (***):

$$(20x_1^2+20x_3^2-160x_3+\tfrac{320x_3}{x_1})p_3^2+(10x_1^2+10x_3^2-80x_3+\tfrac{160x_3}{x_1}+2x_1^2x_3^2-4x_1^3x_3+8x_1^2x_3+(10x_3^2+10x_1^2-80x_3)p_2)p_3+x_1^2x_3^2-2x_1^3x_3+x_1^2x_3=0$$

Теперь для того, чтобы избавиться от слагаемых с p_2p_3 и, следовательно, получить квадратное уравнение относительно p_3, временно сократим на множитель $(x_1 - 2)$, а случай, когда $x_1 = 2$ рассмотрим позже. Получаем систему 2 уравнений с 2 неизвестными. Домножим (****) на $-(10x_3^2+10x_1^2-80x_3)$, а определитель на $-20x_1x_3$ и сложим:

$$(-400x_1^3x_3 - 400x_1x_3^3 + 3200x_1x_3^2 - 6400x_3^2 - 200x_1^4 - 400x_1^2x_3^2 - 200x_3^4 + 1600x_1^2x_3 + 1600x_3^3)p_3^2 + (-200x_1^3x_3 - 200x_1x_3^2 + 1600x_1x_3^2 - 3200x_3^2 - 40x_1^3x_3^3 + 60x_1^4x_3^2 - 160x_1^3x_3^2 - 200x_1^2x_3^2 - 100x_3^4 - 20x_1^2x_3^4 - 100x_1^4 + 800x_1^2x_3 + 800x_3^3 + 160x_1^2x_3^3)p_3 - 20x_1^3x_3^3 + 30x_1^4x_3^2 - 80x_1^3x_3^2 - 10x_1^2x_3^4 + 80x_1^2x_3^3 = 0$$

Так как все коэфициенты зависят только от x_1 и x_3, то, варируя x_1, из уравнения (*) мы получаем x_3, и, следовательно, все коэфициенты квадратного уравнения нам известны. Учитывая, что $p_3 > 0$, решаем уравнение и получаем единсвтенное решение. Для того, чтобы получить для каждого p_3 значение p_2 выразим последнее из уравнения (****):

$$p_2 = \frac{x_3}{2x_1} + \frac{x_1}{2x_3} + \frac{x_1x_3}{10} + \frac{x_3p_3}{x_3} + \frac{x_3p_3}{x_1} + \frac{x_1x_3}{20p_3}$$

Для полученных значений p_2 и p_3 восстановим x_2 по формуле (***) и x_4 по формуле (**).
Варируя x_1 в диапазоне $(0, 2)$, получим первую часть бифуркационной диаграммы.
Для получение второй части бифуркационной диаграммы рассмотрим случай, когда $x_1 = 2$. Тогда из формул (*), (**), (***) получаем $x_1 = x_3 = 2$ и $x_2 = x_4 = \frac{p_2}{2}$. Подставив x_1 и x_3 в определитель, мы получаем значения всех коэфициентов. Выразим p_2 через p_3 и получим:

$$p_2 = 2p_3 + 1.4 + 0.2\frac{1}{p_3}$$

Теперь варируем p_3 и для каждого p_3 получаем значение p_2. Получаем 2 часть бифуркационной диаграммы.

3 Текст программы

3.1 main.cpp

```cpp
#include <iostream>
#include "functions.hpp"

int main() {
  double x1, x2, x3, x4;
  double p2, p3;
  double c1, c2, c3;

  std::cout << "---------------- 1 PART ----------------\n";
  double x = 0.1;
  do {
      x1 = x;
      x3 = 4 - x1;

      calculateC1(c1, x1, x3);
      calculateC2(c2, x1, x3);
      calculateC3(c3, x1, x3);

      calculateP3(p3, c1, c2, c3);
      calculateP2(p2, p3, x1, x3);
      std::cout << "p3: " << p3 << "\n";
      std::cout << "p2: " << p2 << "\n";
      std::cout << '\n';

      calculateX2(x2, p2, p3, x1);
      calculateX4(x4, p3, x1, x2);

      x += 0.1;
  } while (x < 2);

  std::cout << "---------------- 2 PART ----------------\n";
  std::cout << "---- p3 in [0.1, 1.0] with  h = 0.1 ----\n";
  getP2P3(p2, p3, 0.1, 1.0);

  std::cout << "---- p3 in [1, 10] with  h = 1 ----\n";
  getP2P3(p2, p3, 1, 10);

  std::cout << "---- p3 in [10, 100] with  h = 10 ----\n";
  getP2P3(p2, p3, 10, 100);

  std::cout << "----- p3 in [100, 1000] with h = 100 ----\n";
  getP2P3(p2, p3, 100, 1000);

  std::cout << "--THE VALIDATION BLOCK [control point]--\n";
  x1 = 1;
  x3 = 4 - x1;

  calculateC1(c1, x1, x3);
  calculateC2(c2, x1, x3);
  calculateC3(c3, x1, x3);

  calculateP3(p3, c1, c2, c3);
  calculateP2(p2, p3, x1, x3);

  calculateX2(x2, p2, p3, x1);
  calculateX4(x4, p3, x1, x2);

  firstCheck(p2, p3, x1, x2, x3, x4);
  secondAndThirdCheck(p2, p3, x1, x2, x3, x4);

  return 0;
}
```

3.2 functions.hpp

```
void calculateC1(double &c1, double &x1, double &x3);
void calculateC2(double &c2, double &x1, double &x3);
void calculateC3(double &c3, double &x1, double &x3);
void calculateP3(double &p3, double c1, double c2, double c3);
void calculateP2(double &p2, double p3, double x1, double x3);
void calculateX2(double &x2, double p2, double p3, double x1);
void calculateX4(double &x4, double p3, double x1, double x2);
void getP2P3(double &p2, double &p3, double start, double end);
void firstCheck(double p2, double p3, double x1, double x2, double x3, double x4);
void secondAndThirdCheck(double p2, double p3, double x1, double x2, double x3, double x4);
```

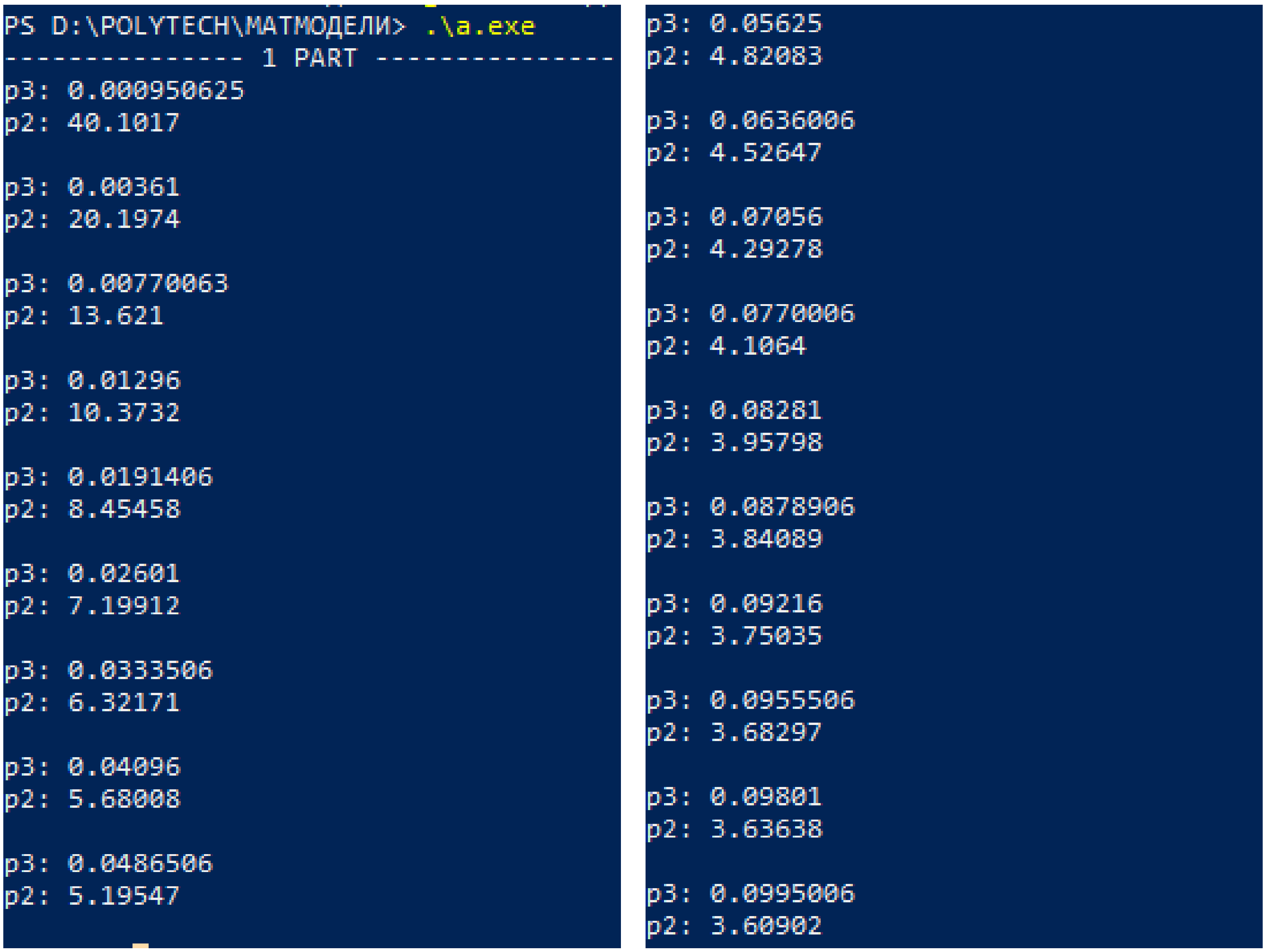
3.3 functions.cpp

```
#include <cmath>
#include <iostream>

#include "cmath.h"
#include "functions.hpp"

void calculateC1(double &c1, double &x1, double &x3) {
  c1 = - 400 * x1 * pow(x3, 3) - 400 * pow(x1, 3) * x3 + 3200 * x1 * pow(x3, 2) - 6400 * pow(x3,
       2) - 200 * pow(x1, 4) - 400 * pow(x1, 2) * pow(x3, 2) - 200 * pow(x3, 4) + 1600 * pow(x1,
       2) * x3 + 1600 * pow(x3, 3);
};

void calculateC2(double &c2, double &x1, double &x3) {
  c2 = - 200 * pow(x1, 3) * x3 - 200 * x1 * pow(x3, 3) + 1600 * x1 * pow(x3, 2) - 3200 * pow(x3,
       2) - 40 * pow(x1, 3) * pow(x3, 3) + 60 * pow(x1, 4) * pow(x3, 2) - 160 * pow(x1, 3) * pow
      (x3, 2) - 200 * pow(x1, 2) * pow(x3, 2) - 100 * pow(x3, 4) - 20 * pow(x1, 2) * pow(x3, 4)
      - 100 * pow(x1, 4) + 800 * pow(x1, 2) * x3 + 800 * pow(x3, 3) + 160 * pow(x1, 2) * pow(x3,
       3);
};

void calculateC3(double &c3, double &x1, double &x3) {
  c3 = - 20 * pow(x1, 3) * pow(x3, 3) + 30 * pow(x1, 4) * pow(x3, 2) - 80 * pow(x1, 3) * pow(x3,
       2) - 10 * pow(x1, 2) * pow(x3, 4) + 80 * pow(x1, 2) * pow(x3, 3);
};

void calculateP2(double &p2, double p3, double x1, double x3) {
  p2 = x3 / (2 * x1) + x1 / (2 * x3) + ( x1 * x3) / 10 + (x1 * p3) / x3
       + (x3 * p3) / x1 + (x1 * x3) / (20 * p3);
};

void calculateP3(double &p3, double c1, double c2, double c3) {
  double d =  pow(c2, 2) - 4 * c1 * c3;
  double p3_1 = (-c2 + sqrt(d)) / (2 * c1);
  double p3_2 = (-c2 - sqrt(d)) / (2 * c1);

  //p3 > 0
  p3 = p3_1 > 0 ? p3_1 : p3_2;
};

//(***)
void calculateX2(double &x2, double p2, double p3, double x1) {
  x2 = (p2 * x1 + (1 +2 * p3) * (x1 - 2)) / (x1 * x1);
};

//(**)
void calculateX4(double &x4, double p3, double x1, double x2) {
  x4 = ((1 + 2 * p3) * (x1 - 2) + 10 * p3 * x2) / (10 * p3);
};

void getP2P3(double &p2, double &p3, double start, double end) {
  p3 = start;
```

```
    while (p3 <= end) {
      p2 = 2 * p3 + 1.4 + 0.2 * (1 / p3);
      std::cout << "p3: " << p3 << "\n";
      std::cout << "p2: " << p2 << "\n";
      std::cout << '\n';
      p3 += start;
    }
    std::cout << '\n';
  };

  void firstCheck(double p2, double p3, double x1, double x2, double x3, double x4) {
    //1 PART
    double dx[4];
    dx[0] = 2 - (p2 + 1) * x1 + pow(x1, 2) * x2 + p3 * (x3 - x1);
    dx[1] = p2 * x1 - pow(x1, 2) * x2 + 10 * p3 * (x4 - x2);
    dx[2] = 2 - (p2 + 1) * x3 + pow(x3, 2) * x4 - p3 * (x3 - x1);
    dx[3] = p2 * x3 - pow(x3, 2) * x4 - 10 * p3 * (x4 - x2);

    bool successfully = true;
    for (int i = 0; i < 4; ++i) {
      if (dx[i] > 1e-15) {
        successfully = false;
        break;
      }
    }

    if (!successfully) {
      std::cout << "1 CHECK FAILED\n";
    } else {
      std::cout << "1 CHECK PASSED\n";
    }
  };

  void secondAndThirdCheck(double p2, double p3, double x1, double x2, double x3, double x4) {
    //2 PART
    int n = 4;
    int nm = 4;
    double wr[n], wi[n];
    int ierr;

    double detY[n][n] = {
      {-p2 -1 + 2 * x1 * x2 - p3, pow(x1, 2), p3, 0},
      {p2 - 2 * x1 * x2, -pow(x1, 2) - 10 * p3, 0, 10 * p3},
      {p3, 0, -p2 - 1 + 2 * x3 * x4 - p3, pow(x3, 2)},
      {0, 10 * p3, p2 - 2 * x3 * x4, -pow(x3, 2) - 10 * p3}
    };

    double matrix[n * n];
    for (int i = 0; i < 4; ++i)
    {
      for (int j = 0; j < 4; ++j)
      {
        matrix[4 * i + j] = detY[i][j];
      }
    }

    qr(n, nm, matrix, wr, wi, &ierr);

    bool successfully = false;
    for (int i = 0; i < n; ++i) {
      if (wr[i] < 1e-15) {
        successfully = true;
        break;
      }
    }

    if (!successfully) {
      std::cout << "2 CHECK FAILED\n";
```

```
    } else {
      std::cout << "2 CHECK PASSED\n";
    }

    //3 PART
    //p2
    std::cout << "3 CHECK: \n";

    detY[0][0] = -x1;
    detY[1][0] = x1;
    detY[2][0] = -x3;
    detY[3][0] = x3;

    for (int i = 0; i < 4; ++i)
    {
      for (int j = 0; j < 4; ++j)
      {
        matrix[4 * i + j] = detY[i][j];
      }
    }

    qr(n, nm, matrix, wr, wi, &ierr);

    for (int i = 0; i < n; ++i) {
      std::cout << "wr: " << wr[i] << " wi: " << wi[i] << '\n';
    }
    std::cout << '\n';

    //p3
    detY[0][0] = x3 - x1;
    detY[1][0] = 10 * (x4 - x2);
    detY[2][0] = x1 - x3;
    detY[3][0] = -10 * (x4 - x2);

    for (int i = 0; i < 4; ++i)
    {
      for (int j = 0; j < 4; ++j)
      {
        matrix[4 * i + j] = detY[i][j];
      }
    }

    qr(n, nm, matrix, wr, wi, &ierr);

    for (int i = 0; i < n; ++i) {
      std::cout << "wr: " << wr[i] << " wi: " << wi[i] << '\n';
    }
    std::cout << '\n';
  };
```

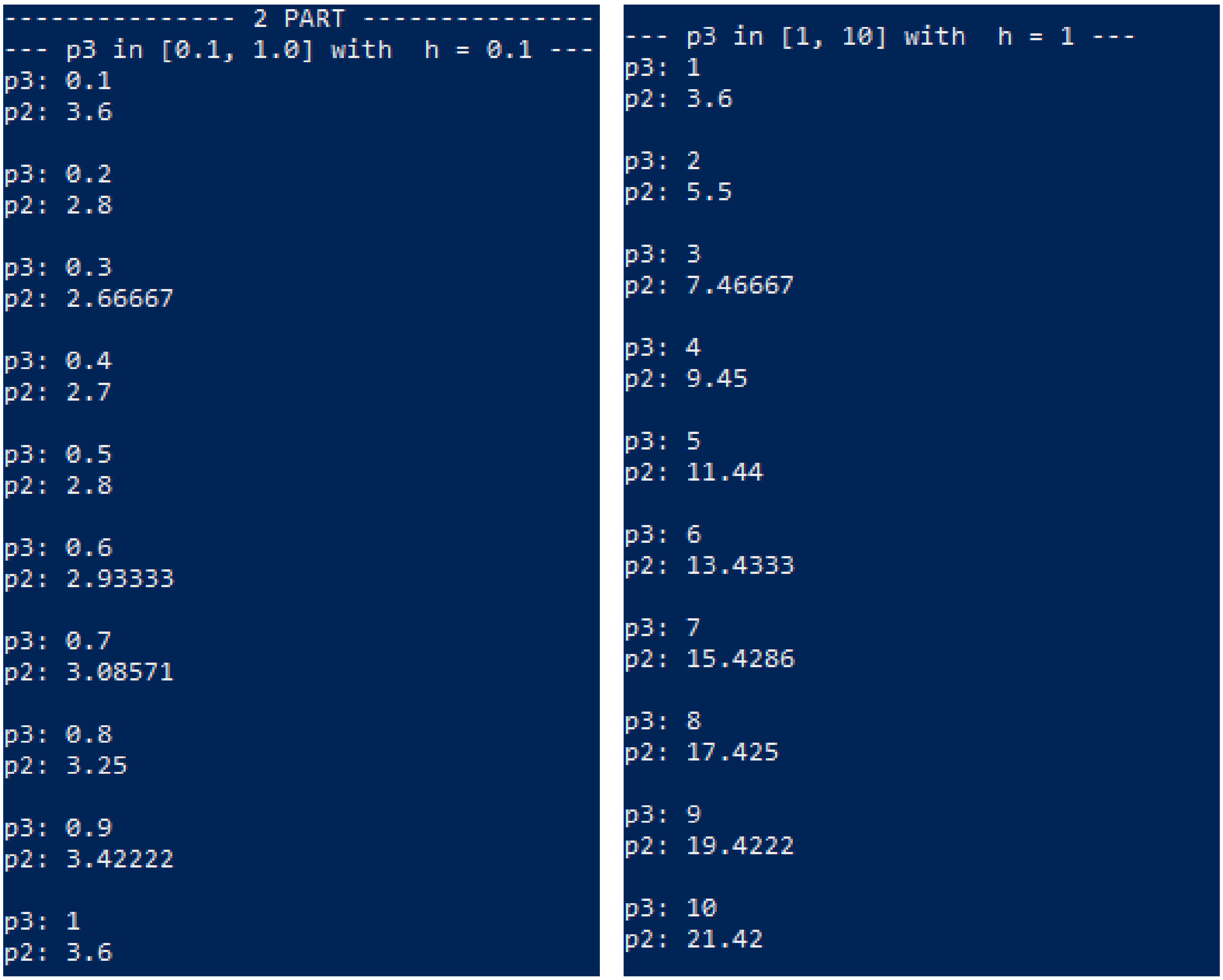
4 Результаты



```
PS D:\POLYTECH\МАТМОДЕЛИ> .\a.exe
---------------- 1 PART ----------------
p3: 0.000950625
p2: 40.1017

p3: 0.00361
p2: 20.1974

p3: 0.00770063
p2: 13.621

p3: 0.01296
p2: 10.3732

p3: 0.0191406
p2: 8.45458

p3: 0.02601
p2: 7.19912

p3: 0.0333506
p2: 6.32171

p3: 0.04096
p2: 5.68008

p3: 0.0486506
p2: 5.19547
```

```
p3: 0.05625
p2: 4.82083

p3: 0.0636006
p2: 4.52647

p3: 0.07056
p2: 4.29278

p3: 0.0770006
p2: 4.1064

p3: 0.08281
p2: 3.95798

p3: 0.0878906
p2: 3.84089

p3: 0.09216
p2: 3.75035

p3: 0.0955506
p2: 3.68297

p3: 0.09801
p2: 3.63638

p3: 0.0995006
p2: 3.60902
```

Figure 1: 1 часть бифуркационной диаграммы



```
--------------- 2 PART ---------------
--- p3 in [0.1, 1.0] with  h = 0.1 ---
p3: 0.1
p2: 3.6

p3: 0.2
p2: 2.8

p3: 0.3
p2: 2.66667

p3: 0.4
p2: 2.7

p3: 0.5
p2: 2.8

p3: 0.6
p2: 2.93333

p3: 0.7
p2: 3.08571

p3: 0.8
p2: 3.25

p3: 0.9
p2: 3.42222

p3: 1
p2: 3.6
```

```
--- p3 in [1, 10] with  h = 1 ---
p3: 1
p2: 3.6

p3: 2
p2: 5.5

p3: 3
p2: 7.46667

p3: 4
p2: 9.45

p3: 5
p2: 11.44

p3: 6
p2: 13.4333

p3: 7
p2: 15.4286

p3: 8
p2: 17.425

p3: 9
p2: 19.4222

p3: 10
p2: 21.42
```

Figure 2: 2.1 часть бифуркационной диаграммы

```
--- p3 in [10, 100] with  h = 10 ---
p3: 10
p2: 21.42

p3: 20
p2: 41.41

p3: 30
p2: 61.4067

p3: 40
p2: 81.405

p3: 50
p2: 101.404

p3: 60
p2: 121.403

p3: 70
p2: 141.403

p3: 80
p2: 161.403

p3: 90
p2: 181.402

p3: 100
p2: 201.402
```

```
---- p3 in [100, 1000] with h = 100 --
p3: 100
p2: 201.402

p3: 200
p2: 401.401

p3: 300
p2: 601.401

p3: 400
p2: 801.4

p3: 500
p2: 1001.4

p3: 600
p2: 1201.4

p3: 700
p2: 1401.4

p3: 800
p2: 1601.4

p3: 900
p2: 1801.4

p3: 1000
p2: 2001.4
```

Figure 3: 2.2 часть бифуркационной диаграммы

```
-THE VALIDATION BLOCK [control point]-
1 CHECK PASSED
2 CHECK PASSED
3 CHECK:
wr: -0.202391 wi: 0
wr: -2.25052 wi: 0.809417
wr: -2.25052 wi: -0.809417
wr: -2.91532 wi: 0

wr: 0.313763 wi: 3.64687
wr: 0.313763 wi: -3.64687
wr: -2.62314 wi: 2.05462
wr: -2.62314 wi: -2.05462
```

Figure 4: Блок проверки

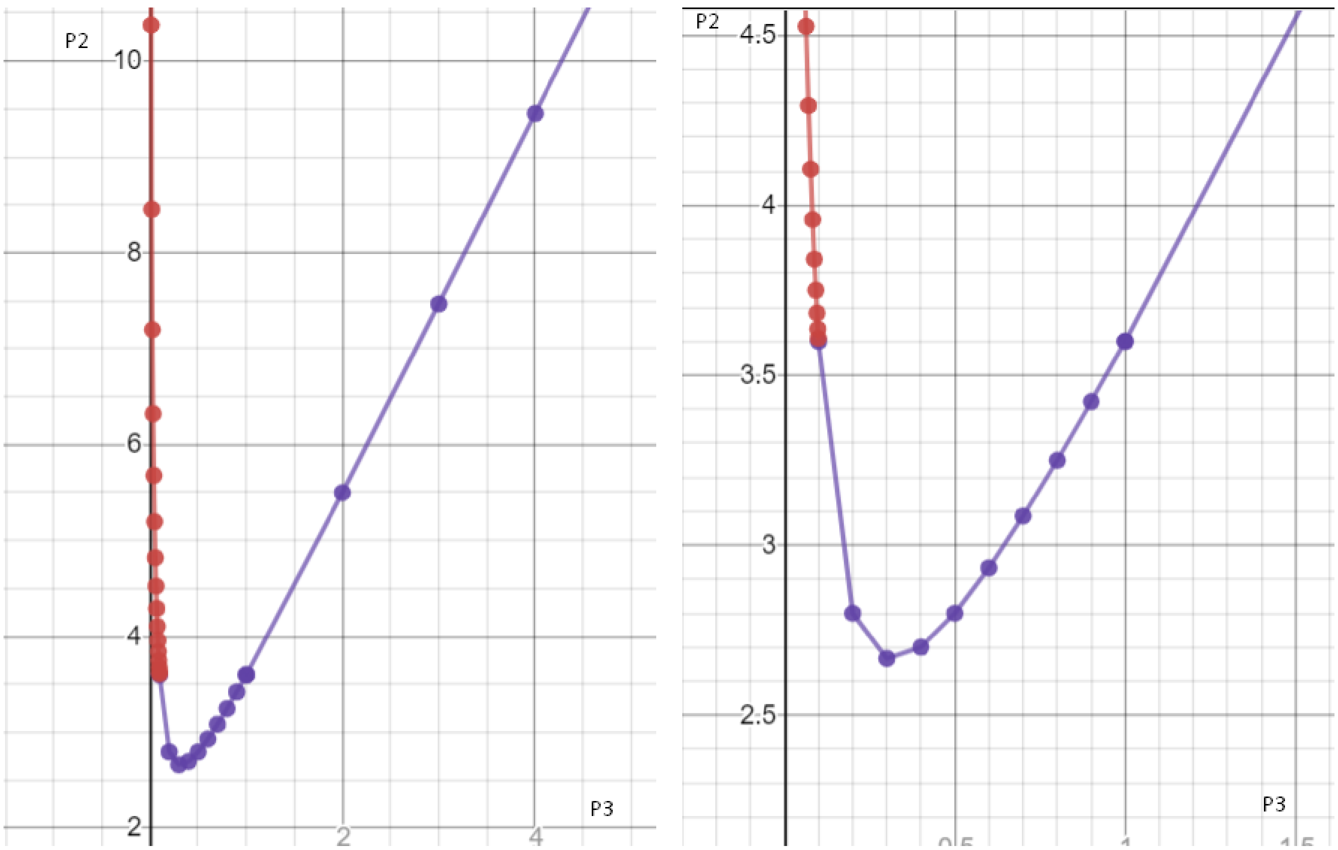


Figure 5: Бифуркационая диаграмма

5 Блок проверки

Блок проверки состоит из 3-х частей:
`Часть 1:` проверка стационарных точек. Для этого подставим нашу точку в исходные уравнения. Теоритически должен быть нуль, но фактически получаем очень близкие к нулю значения.
`Часть 2:` необходимо показать, что определитель равен нулю. Для этого берем начальную матрицу Якоби и делаем ее числовой, подставив значения. Затем рассчитываем собственные значения матрицы с помощью qr-алгоритма. Одно из значений должно быть нулевым (очень близким к нулю).
`Часть 3:` проверка на поворот и ветвление. Вычеркиваем первый столбец и заменяем его на столбец частных производных по параметру p_2. Делаем матрицу числовой. Проверяем собственные значения. Тоже самое повторяем с параметром p_3. Так как получили ненулевые значения, следовательно, это точка поворота.
Для первой части блока проверки была написана функция `firstCheck()`, а для второй и третьей части - `secondAndThirdCheck()`. Код программы и результаты представлены выше.

6 Выводы

Для исходной модели была построена бифуркационная диаграмма точек вещественной бифуркации в плоскости параметров (p_3, p_2). Для этого сначала были проведены необходимые аналитические преобразования, описанные в соответсвующем блоке. Затем написана программная реализация, результатом которой являются необходимые для построения точки. А также был успешно пройден блок проверки. В первой части мы проверили, что наша точка действительно является стационарной. Во второй части мы доказали, что определитель равен нулю, так как одно из собственных значений равно нулю. И в заключительной части мы проверили точку на поворот и ветвление. Так как ни одно из полученных собственных значений матрицы не является нулевым, следовательно, имеем дело с точкой поворота.