Министерство науки и высшего образования Российской Федерации

Новосибирский государственный технический университет

Кафедра автоматизированных систем управления

Лабораторная работа №5

«Расчет переходной функции численными методами»

Группа: АВТ-813

Студент:

Чернаков Кирилл

Преподаватель:

Достовалов Дмитрий Николаевич,

к.т.н., заведующий кафедрой АСУ

Новосибирск

2020 г.

1. Передаточная функция

$$W(p) = \frac{0\mathrm{p}^2 - 0.23\mathrm{p} + 0.99}{1.11\mathrm{p}^2 + 0.98\mathrm{p} - 1.19}$$

ДУ:

$$1.11\mathrm{y}''(\mathrm{x}) + 0.98\mathrm{y}'(\mathrm{x}) - 1.19y(x) = 0\mathrm{g}''(\mathrm{x}) - 0.23\mathrm{g}'(\mathrm{x}) + 0.99\mathrm{g}(\mathrm{x})$$

2. Структурная схема в Matlab

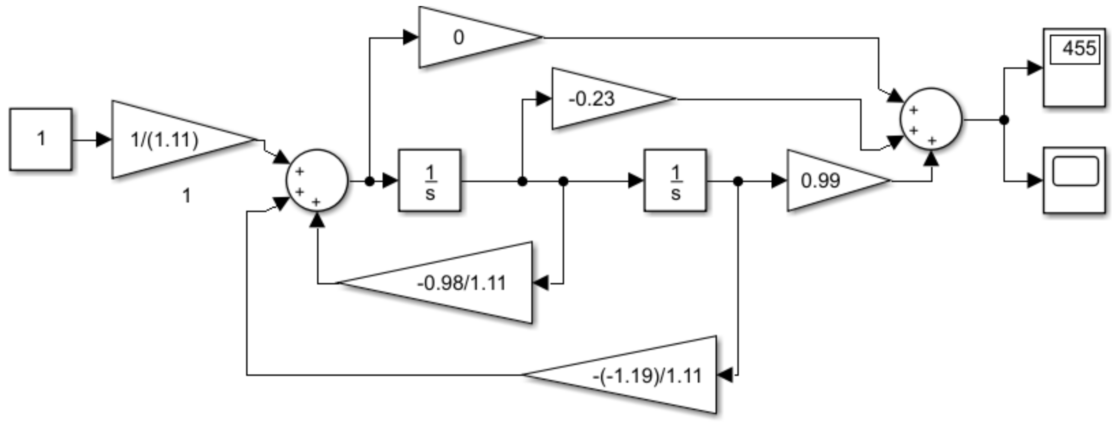

Рис. 1 – Структурная схема

3. График переходной характеристики из Matlab

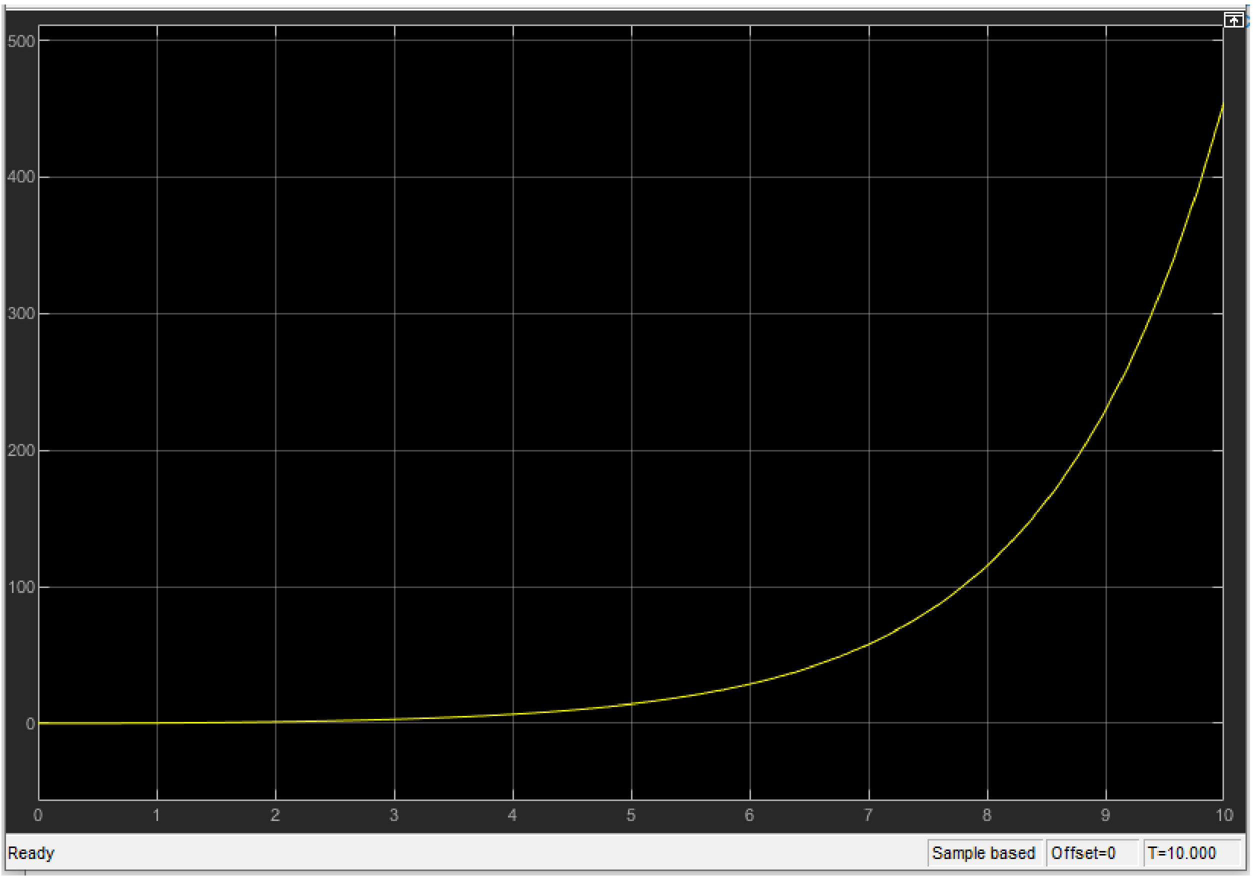

Рис. 2 – График переходной функции

4. Значения y и t_k, полученные в Matlab.

Проверим эту систему на устойчивость, рассмотри характеристическое уравнение системы:

$$1.11p^2 + 0.98p - 1.19 = 0$$

Найдем корни:

$$p_1 = -1.56703$$

$$p_1 = 0.684144$$

Так как не все корни характеристического уравнения имеют отрицательную вещественную часть, следует что система не устойчивая и она никогда не придёт к установившемуся режиму.

Рассмотрим переходной процесс за время $\mathbf{t}_{\boldsymbol{k}}$ **= 10**, за это время процесс придёт к значению **y = 455.97571918554**

5. Система уравнений, используемая для расчета переходного процесса.

$$\begin{cases} x_1' = x_2 \\ x_2' = x_3 \\ x_3 = \frac{1}{1.11}g + \frac{0.98}{1.11}x_2 - \frac{1.19}{1.11}x_1 \\ y = 0.99x_1 - 0.23x_2 + 0x_3 \end{cases}$$

6. Значения y, полученные с помощью ваших программ.

Полусонное значение с помощью программы (Метод Эйлера):

y_1= 453.91070879070827

Полусонное значение с помощью программы (Метод РунгеКутты):

y_1= 456.8335869802985

Значение, полученное с **помощью** Matlab: $\boldsymbol{y_2}$ **= 455.97571918554**

Абсолютная погрешность для Эйлера:

$$\partial = y_2 - y_1$$
$$\partial = 2.06501039483$$

Относительная погрешность для Эйлера:

$$\Delta = \frac{\partial}{y_2} \cdot 100\% = 0.452877271298\%$$

Абсолютная погрешность для Рунге-Кутты:

$$\partial = y_2 - y_1$$
$$\partial = -0.857867794758$$

Относительная погрешность для Рунге-Кутты:

$$\Delta = \frac{\partial}{y_2} \cdot 100\% = 0.188138920268\%$$

Выводы об оценке точности расчетов:

В результате относительная погрешность равняется 0.6398%, что является незначительной ошибкой, из этого следует правильность работы нашей программы и составленной системы уравнений.

Как можно увидеть у метода Рунге-Кутты относительная погрешность меньше, чем у метода Эйлера, что говорит о высокой точности метода Рунге-Кутты.

7. Скриншоты из программы

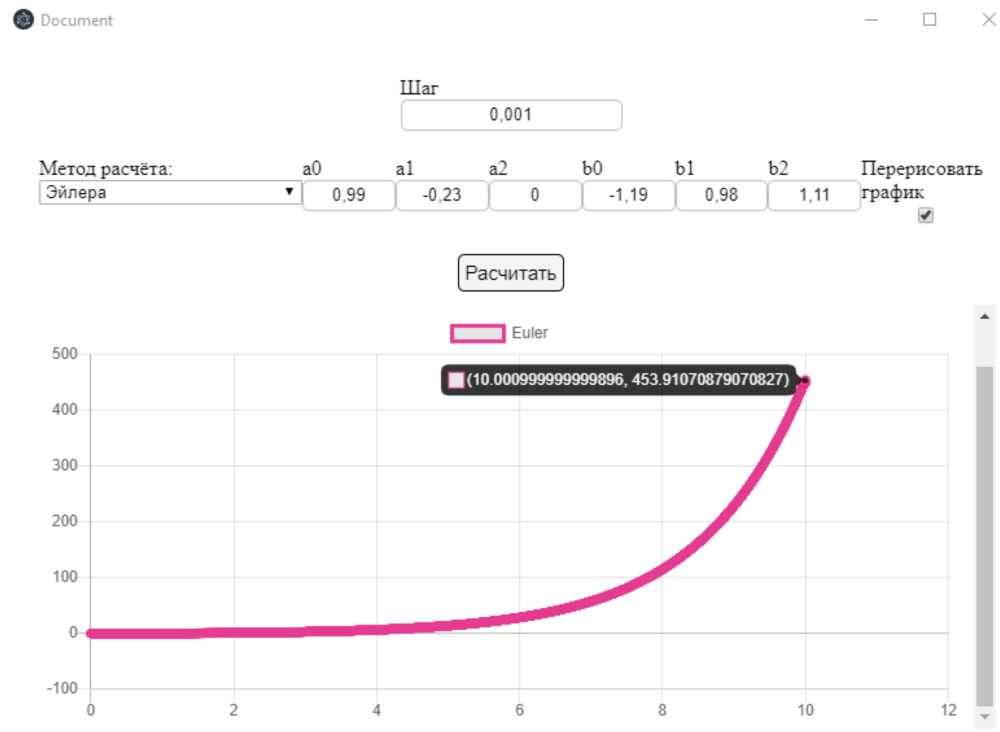

Рис. 3 – Скриншот работы программы

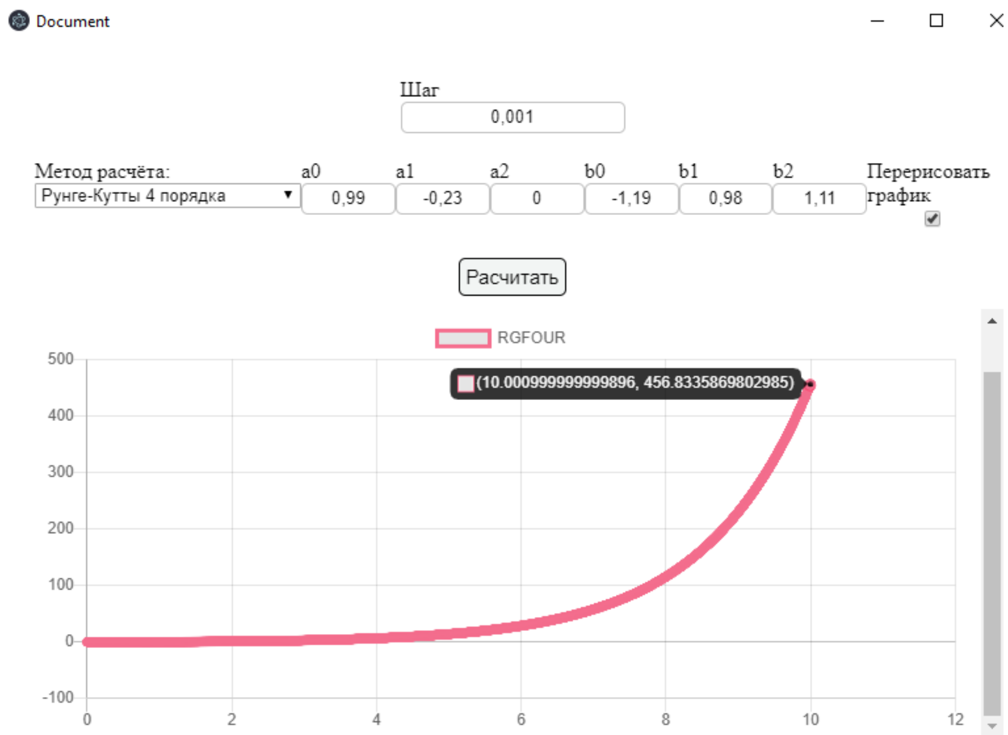

Рис. 4 – Скриншот работы программы

8. Материалы по дополнительным заданиям

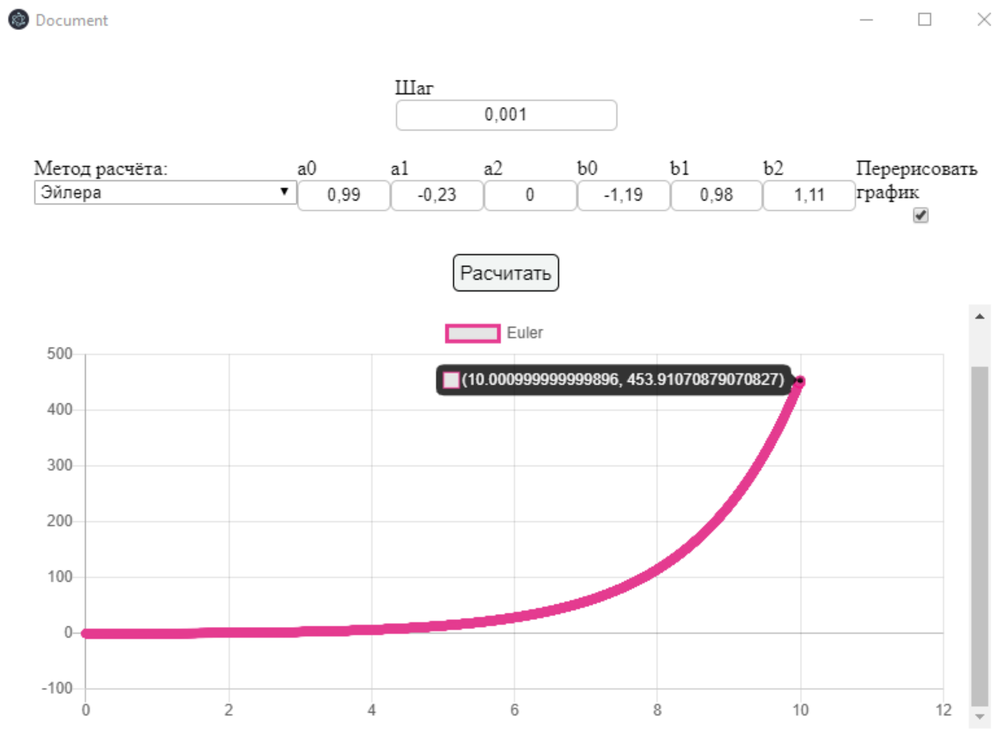

Рис. 5 – График переходной функции

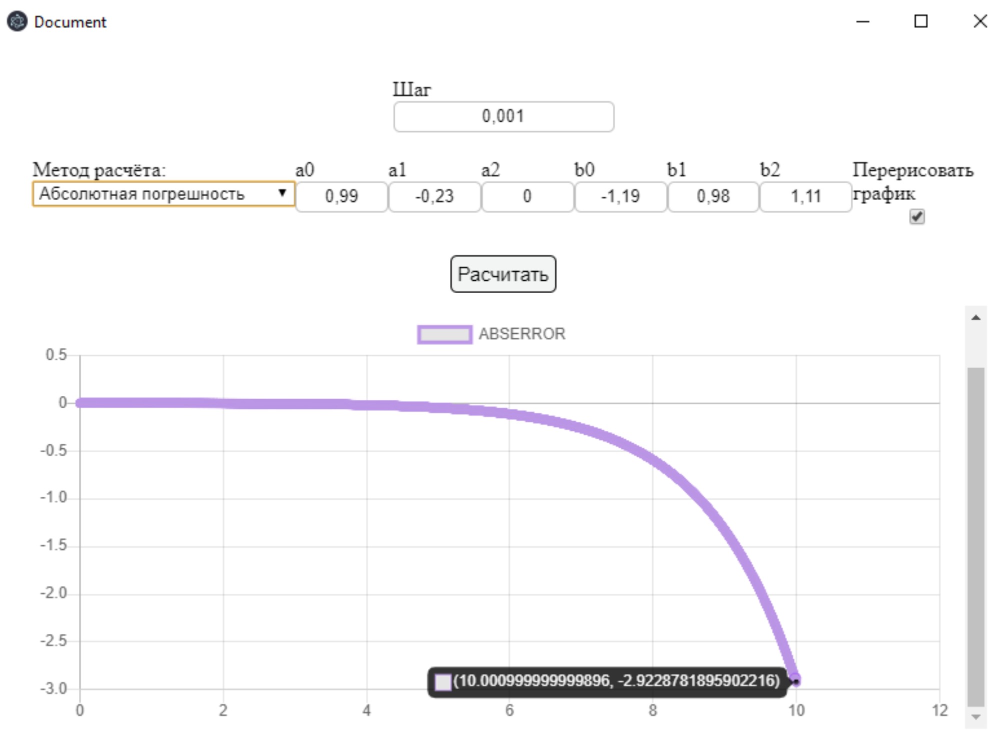

Рис. 6 – График абсолютной погрешности для метода Эйлера и Рунге-Кутты

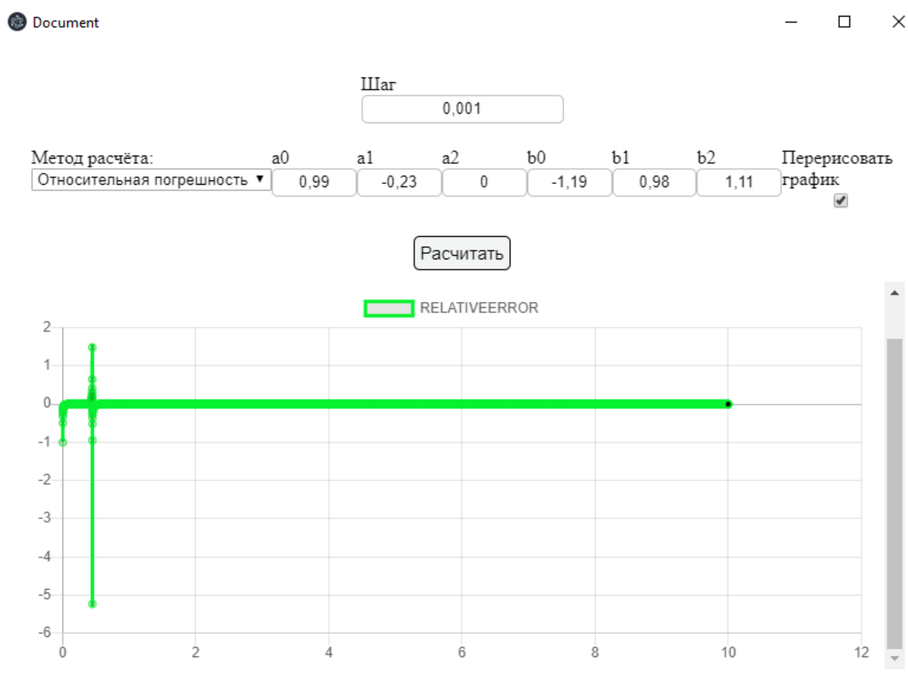

Рис. 7 – График относительной погрешности для метода Эйлера и Рунге-Кутты

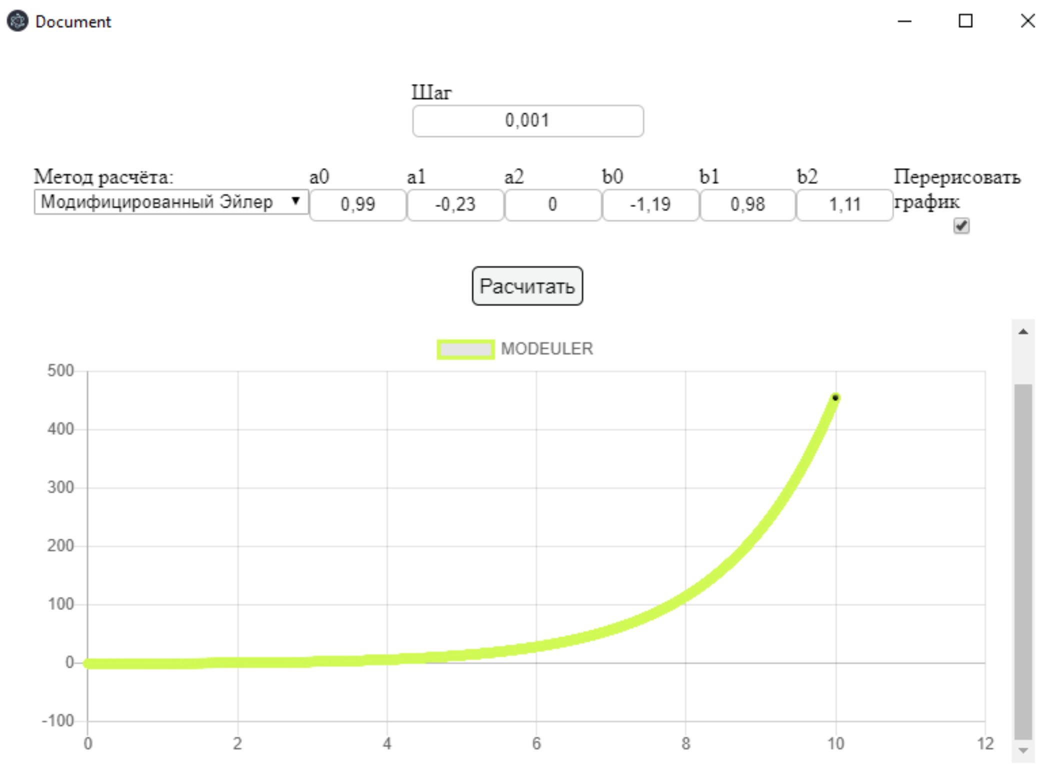

Рис. 8 – Скриншот работы программы

Модифицированный метод Эйлера

Полусонное значение с помощью программы(Метод Мод. Эйлера):
$\boldsymbol{y}_{1}$**= 455.4678522534885**

Значение полученное с помощью Matlab: $\boldsymbol{y_2}$ **= 455.97571918554**

Абсолютная погрешность:

$$\partial = y_2 - y_1$$
$$\partial = 0{,}1351$$

Относительная погрешность:

$$\Delta = \left|\frac{\partial}{y_2} \cdot 100\%\right| = 0.111380257914\%$$

Выводы: в результате мы добились уменьшение ошибки по сравнению с Эйлером, путем изменения метода численного решения дифференциального уравнения.

9. Приложение

```
export class DiffEquations
{
    constructor( coefs )
    {
        this.coefs = coefs
    }

    absoluteError( step ) {
        let firstResults = this.euler( step )
        let secondResults = this.rgFour( step )
        let results = []
        for( const [ index, result ] of firstResults.entries() ) {
            results.push( { x: result.x, y: result.y - secondResults[ index ].y } )
        }

        return results
    }

    relativeError( step ) {
```

```
        let absoluteErrorResults = this.absoluteError( step )
        let secondResults = this.rgFour( step )
        let results = []
        for( const [ index, result ] of absoluteErrorResults.entries() ) {
            results.push( { x: result.x, y: result.y / secondResults[ index ].y } )
        }

        return results
    }

    euler( step ) {
        let x1 = 0, x2 = 0, x3 = 0, y = 0, x = 0
        let results = []

        for( let i = 0; i < 10001; i++ ) {
            x1 += step * x2
            x2 += step * x3
            x3 = 1 / this.coefs.b2 - this.coefs.b1 / this.coefs.b2 * x2 - this.coefs.b0 / this.coefs.b2 * x1

            y = this.coefs.a0 * x1 + this.coefs.a1 * x2 + this.coefs.a2 * x3
            x += step
            results.push( { x, y } )
        }
        return results
    }

    modEuler( step ) {
        let x1 = 0, x2 = 0, x3 = 0, y = 0, x = 0
        let results = []

        for( let i = 0; i < 10001; i++ ) {
            x1 += step * ( x2 + step / 2 * x2 )
            x2 += step * ( x3 + step / 2 * x3 )
            x3 = 1 / this.coefs.b2 - this.coefs.b1 / this.coefs.b2 * x2 - this.coefs.b0 / this.coefs.b2 * x1

            y = this.coefs.a0 * x1 + this.coefs.a1 * x2 + this.coefs.a2 * x3
            x += step
            results.push( { x, y } )
        }
        return results
    }

    rgFour( step ) {
        let x1 = 0, x2 = 0, y = 0, x = 0
        let k1 = 0, k2 = 0, k3 = 0, k4 = 0, m1 =0, m2 = 0, m3 = 0, m4 =0
```

```

        let results = []

        for ( let i = 0; i < 10001; i++ ) {
            m1 = x2
            m2 = x2 + step / 2 * m1
            m3 = x2 + step / 2 * m2
            m4 = x2 + step * m3

            x1 += ( m1 + 2 * m2 + 2 * m3 + m4 ) * step / 6

            k1 = ( -this.coefs.b0 * x1 - this.coefs.b1 * x2 + 1 ) / this.coefs.b2
            k2 =( -this.coefs.b0 * ( x1 + step / 2 * m1 ) - this.coefs.b1 * ( x2 + step * k1 / 2 ) + 1 ) / this.coefs.b2
            k3 = ( -this.coefs.b0 * ( x1 + step / 2 * m2 ) - this.coefs.b1 * ( x2 + step * k2 / 2 ) + 1 ) / this.coefs.b2
            k4 = ( -this.coefs.b0 * ( x1 + step * m3 ) - this.coefs.b1 * ( x2 + step * k3 ) + 1 ) / this.coefs.b2

            x2 += ( k1 + 2 * k2 + 2 * k3 + k4 ) * step / 6
            y = this.coefs.a0 * x1 + this.coefs.a1 * x2 + this.coefs.a2 * ( -this.coefs.b0 * x1 - this.coefs.b1 * x2 + 1 ) / this.coefs.b2

            x += step

            results.push( { x, y } )
        }
        return results
    }
}

```