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**

**=**

1. **Cтруктурная схема в Matlab**

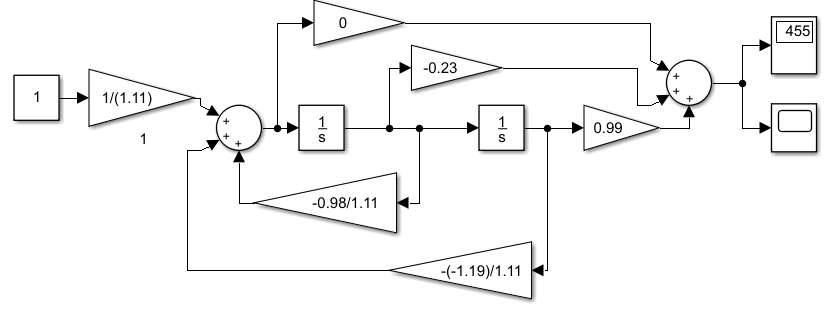
****

Рис. 1 – Структурная схема

1. **График переходной характеристики из Matlab**

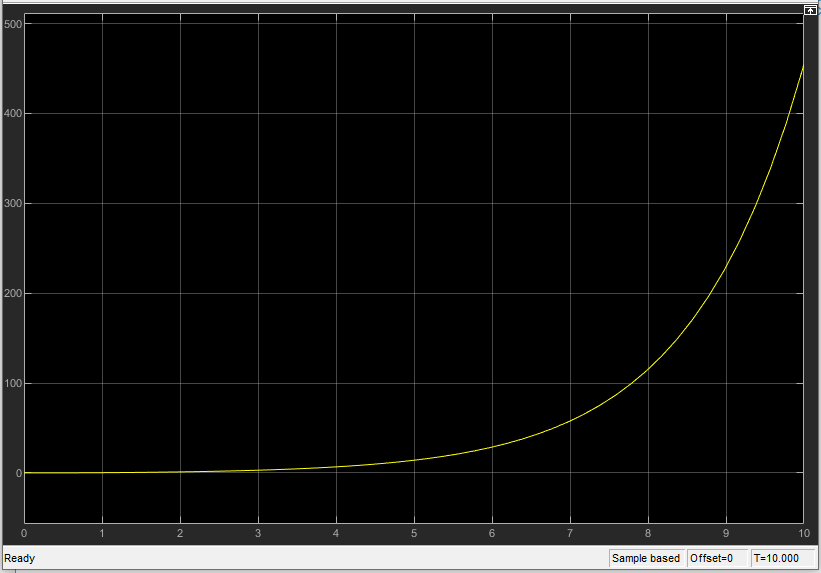
****

Рис. 2 – График переходной функции

1. **Значения y и, полученные в Matlab.**

Проверим эту систему на устойчивость, рассмотри характеристическое уравнение системы:

Найдем корни:

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  **= 10**, за это время процесс придёт к значению **y = 455.97571918554**

1. **Система уравнений, используемая для расчета переходного процесса.**
2. **Значения y, полученные с помощью ваших программ.**

Полусонное значение с помощью программы (Метод Эйлера):

= 453.91070879070827

Полусонное значение с помощью программы (Метод РунгеКутты):

= 456.8335869802985

Значение, полученное с **помощью** Matlab: **= 455.97571918554**

**Абсолютная погрешность для Эйлера:**

-

2.06501039483

**Относительная погрешность для** **Эйлера:**

=0.452877271298%

**Абсолютная погрешность для Рунге-Кутты:**

-

-0.857867794758

**Относительная погрешность для** **Рунге-Кутты:**

=0.188138920268%

**Выводы об оценке точности расчетов:**

В результатеотносительнаяпогрешность 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.

1. **Скриншоты из программы**

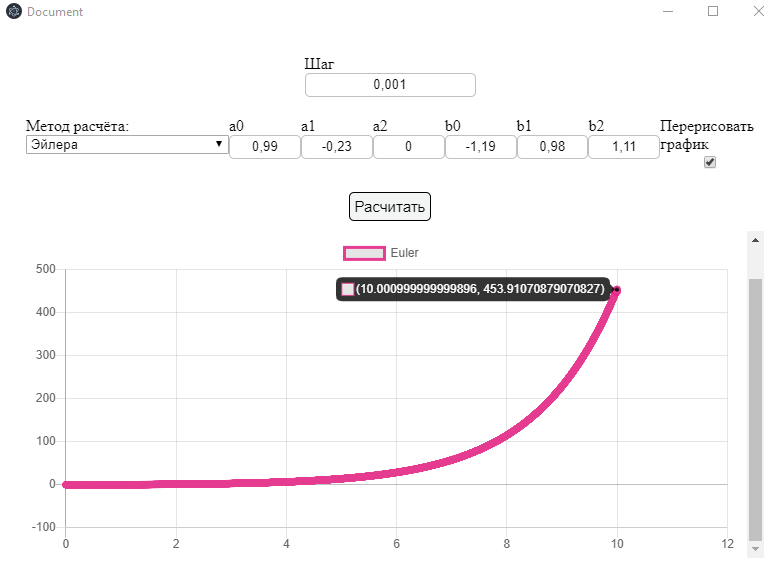
****

Рис. 3 – Скриншот работы программы

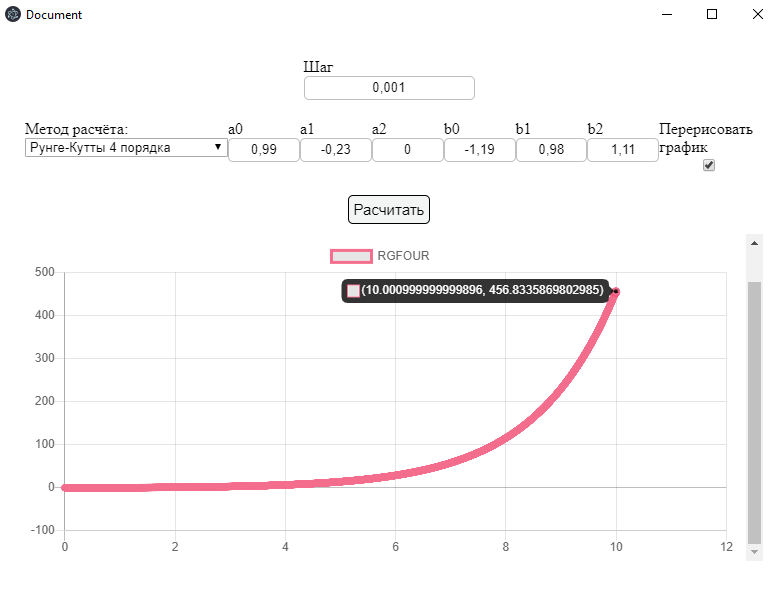


Рис. 4 – Скриншот работы программы

1. **Материалы по дополнительным заданиям**

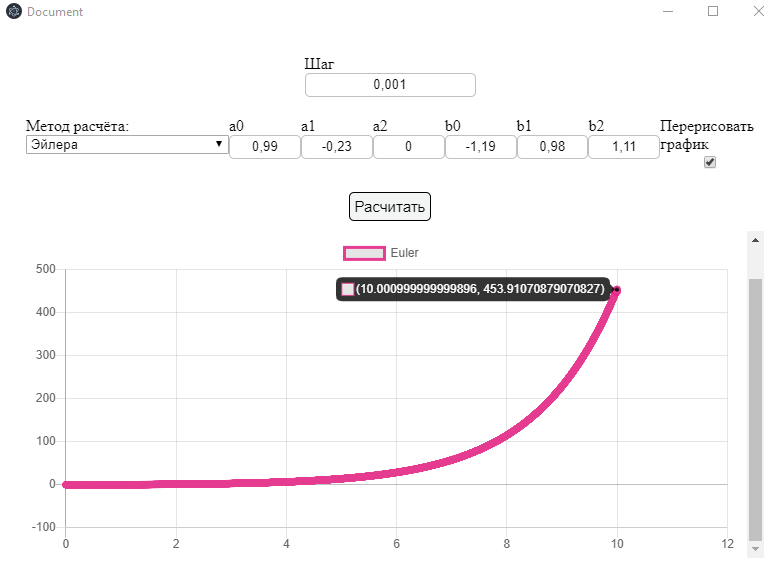
****

Рис. 5 – График переходной функции

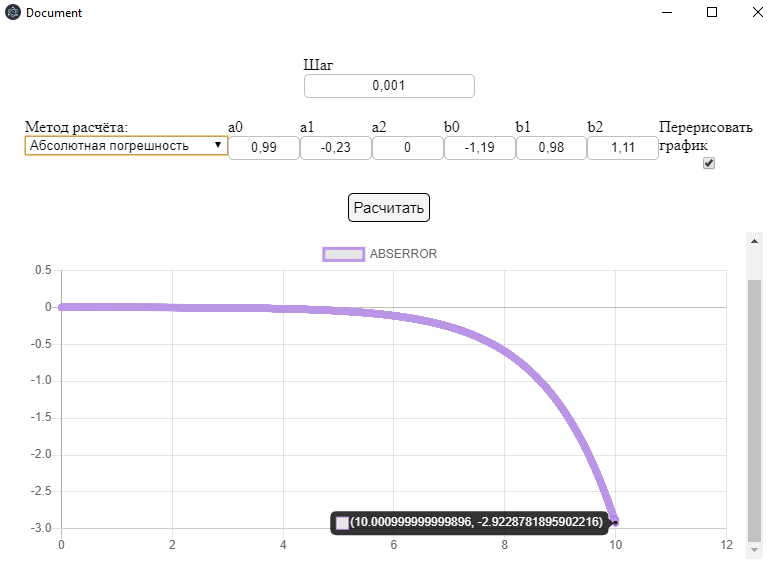
****

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

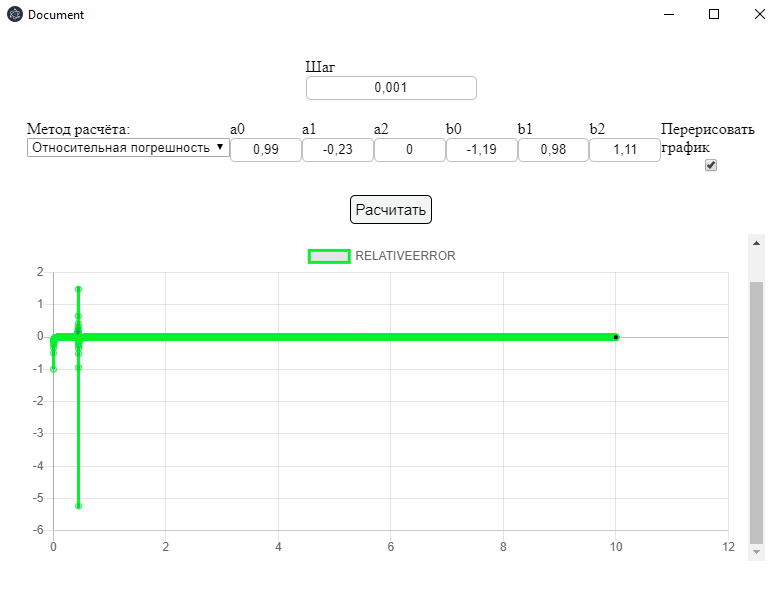
****

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

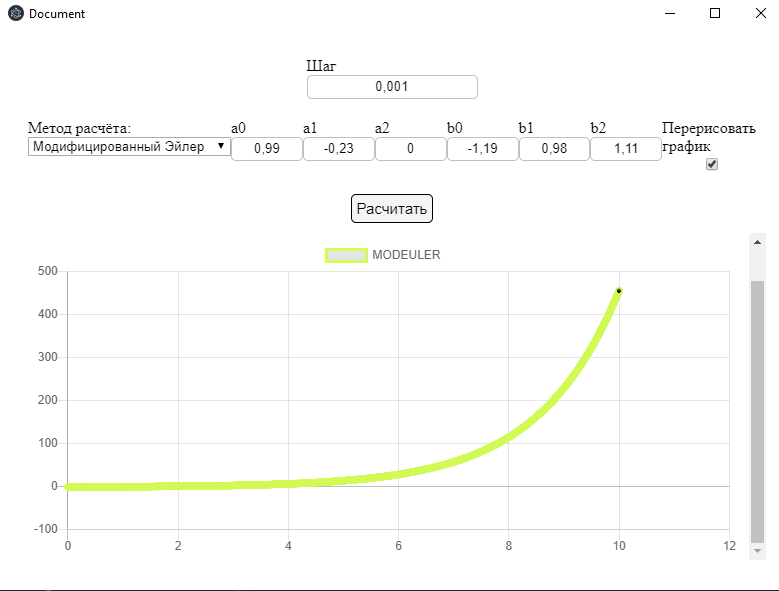
****

Рис. 8 – Скриншот работы программы

**Модифицированный метод Эйлера**

Полусонное значение с помощью программы(Метод Мод. Эйлера ): **= 455.4678522534885**

Значение полученное с помощью Matlab: **= 455.97571918554**

**Абсолютная погрешность:**

-

*0,1351*

**Относительная погрешность:**

=0.111380257914%

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

1. **Приложение**
2. export class DiffEquations
3. {
4. constructor( *coefs* )
5. {
6. this.coefs = *coefs*
7. }
8. absoluteError( *step* ) {
9. let firstResults = this.euler( *step* )
10. let secondResults = this.rgFour( *step* )
11. let results = []
12. for( const [ index, result ] of firstResults.entries() ) {
13. results.push( { x: result.x, y: result.y - secondResults[ index ].y } )
14. }
15. return results
16. }
17. relativeError( *step* ) {
18. let absoluteErrorResults = this.absoluteError( *step* )
19. let secondResults = this.rgFour( *step* )
20. let results = []
21. for( const [ index, result ] of absoluteErrorResults.entries() ) {
22. results.push( { x: result.x, y: result.y / secondResults[ index ].y } )
23. }
24. return results
25. }
26. euler( *step* ) {
27. let x1 = 0, x2 = 0, x3 = 0, y = 0, x = 0
28. let results = []
29. for( let i = 0; i < 10001; i++ ) {
30. x1 += *step* \* x2
31. x2 += *step* \* x3
32. x3 = 1 / this.coefs.b2 - this.coefs.b1 / this.coefs.b2 \* x2 - this.coefs.b0 / this.coefs.b2 \* x1
34. y = this.coefs.a0 \* x1 + this.coefs.a1 \* x2 + this.coefs.a2 \* x3
35. x += *step*
36. results.push( { x, y } )
37. }
38. return results
39. }
40. modEuler( *step* ) {
41. let x1 = 0, x2 = 0, x3 = 0, y = 0, x = 0
42. let results = []
43. for( let i = 0; i < 10001; i++ ) {
44. x1 += *step* \* ( x2 + *step* / 2 \* x2 )
45. x2 += *step* \* ( x3 + *step* / 2 \* x3 )
46. x3 = 1 / this.coefs.b2 - this.coefs.b1 / this.coefs.b2 \* x2 - this.coefs.b0 / this.coefs.b2 \* x1
47. y = this.coefs.a0 \* x1 + this.coefs.a1 \* x2 + this.coefs.a2 \* x3
48. x += *step*
49. results.push( { x, y } )
50. }
51. return results
52. }
53. rgFour( *step* ) {
54. let x1 = 0, x2 = 0, y = 0, x = 0
55. let k1 = 0, k2 = 0, k3 = 0, k4 = 0, m1 =0, m2 = 0, m3 = 0, m4 =0
56. let results = []
57. for ( let i = 0; i < 10001; i++ ) {
58. m1 = x2
59. m2 = x2 + *step* / 2 \* m1
60. m3 = x2 + *step* / 2 \* m2
61. m4 = x2 + *step* \* m3
62. x1 += ( m1 + 2 \* m2 + 2 \* m3 + m4 ) \* *step* / 6
63. k1 = ( -this.coefs.b0 \* x1 - this.coefs.b1 \* x2 + 1 ) / this.coefs.b2
64. k2 =( -this.coefs.b0 \* ( x1 + *step* / 2 \* m1 ) -this.coefs.b1 \* ( x2 + *step* \* k1 / 2 ) + 1 ) / this.coefs.b2
65. k3 = ( -this.coefs.b0 \* ( x1 + *step* / 2 \* m2 ) -this.coefs.b1 \* ( x2 + *step* \* k2 / 2 ) + 1 ) / this.coefs.b2
66. k4 = ( -this.coefs.b0 \* ( x1 + *step* \* m3 ) -this.coefs.b1 \* ( x2 + *step* \* k3 ) + 1 ) / this.coefs.b2
67. x2 += ( k1 + 2 \* k2 + 2 \* k3 + k4 ) \* *step* / 6
68. y = this.coefs.a0 \* x1 + this.coefs.a1 \* x2 + this.coefs.a2 \* ( -this.coefs.b0 \* x1 - this.coefs.b1 \* x2 + 1 ) / this.coefs.b2
69. x += *step*
70. results.push( { x, y } )
71. }
72. return results
73. }
74. }