Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

Высшего образования

«Национальный исследовательский университет ИТМО»

Факультет Программной Инженерии и Компьютерной Техники

Лабораторная работа 6 по вычислительной математике

Численное решение обыкновенных дифференциальных уравнений

Вариант №10

Преподаватель: Малышева Татьяна Алексеева

Выполнил: Состанов Тимур Айратович

Группа: Р3214

Г. Санкт-Петербург

2024

Оглавление

Цель работы 3
Порядок выполнения работы 4
Рабочие формулы 6
Программная реализация задачи 10
Вывод 20

Цель работы

Решить задачу Коши для обыкновенных дифференциальных уравнений численными методами

Порядок выполнения работы

1. В программе численные методы решения обыкновенных дифференциальных уравнений (ОДУ) должен быть реализован в виде отдельного класса /метода/функции;

2. Пользователь выбирает ОДУ вида (не менее трех уравнений), из тех, которые предлагает программа;

3. Предусмотреть ввод исходных данных с клавиатуры: начальные условия $y_0 = y(x_0)$, интервал дифференцирования $[x_0, x_n]$, шаг h, точность ε;

4. Для исследования использовать одношаговые методы и многошаговые методы (см. табл.1);

<u>Одношаговые методы:</u>

1. Метод Эйлера,
2. Усовершенствованный метод Эйлера,
3. Метод Рунге-Кутта 4- го порядка.

<u>Многошаговые методы:</u>

4. Адамса,
5. Милна.

Таблица 1. Варианты задания для программной реализации задачи

№ варианта	Метод	№ варианта	Метод
1	1, 3, 4	16	1, 3, 5
2	2, 3, 5	17	1, 2, 4
3	1, 3, 5	18	1, 3, 5
4	1, 2, 4	19	1, 3, 4
5	2, 3, 5	20	2, 3, 5
6	1, 3, 4	21	1, 3, 4
7	1, 2, 5	22	1, 2, 5
8	2, 3, 4	23	2, 3, 4
9	1, 2, 5	24	1, 3, 4
10	1, 3, 5	25	1, 3, 5
11	2, 3, 4	26	2, 2, 5
12	1, 3, 5	27	1, 3, 4
13	1, 2, 5	28	1, 3, 5
14	2, 3, 5	29	2, 3, 5
15	1, 3, 4	30	1, 2, 4

5. Составить таблицу приближенных значений интеграла дифференциального уравнения, удовлетворяющего начальным условиям, для всех методов, реализуемых в программе;

6. Для оценки точности одношаговых методов использовать правило Рунге;

7. Для оценки точности многошаговых методов использовать точное решение задачи:

$\varepsilon = max_{0 \le i \le n} |y_{i\ \text{точн}} - y_i|$;

8. Построить графики точного решения и полученного приближенного решения (разными цветами);
9. Программа должна быть протестирована при различных наборах данных, в том числе и некорректных.

10. Проанализировать результаты работы программы.

Рабочие формулы

Введем последовательность равноотстоящих точек $x_0, x_1, \ldots, x_n$ (узлов), выбрав малый шаг $h = x_{i+1} - x_i = const$. Тогда получаем **формулу Эйлера**:

$$\boldsymbol{y_{i+1} = y_i + hf(x_i, y_i)} \qquad (7)$$

При i=0 находим значение сеточной функции y_1 при $x = x_1$:

$$y_1 = y_0 + hf(x_0, y_0)$$

Значение y_0 задано начальным условием:

$$\boldsymbol{y_0 = Y_0} \qquad (8)$$

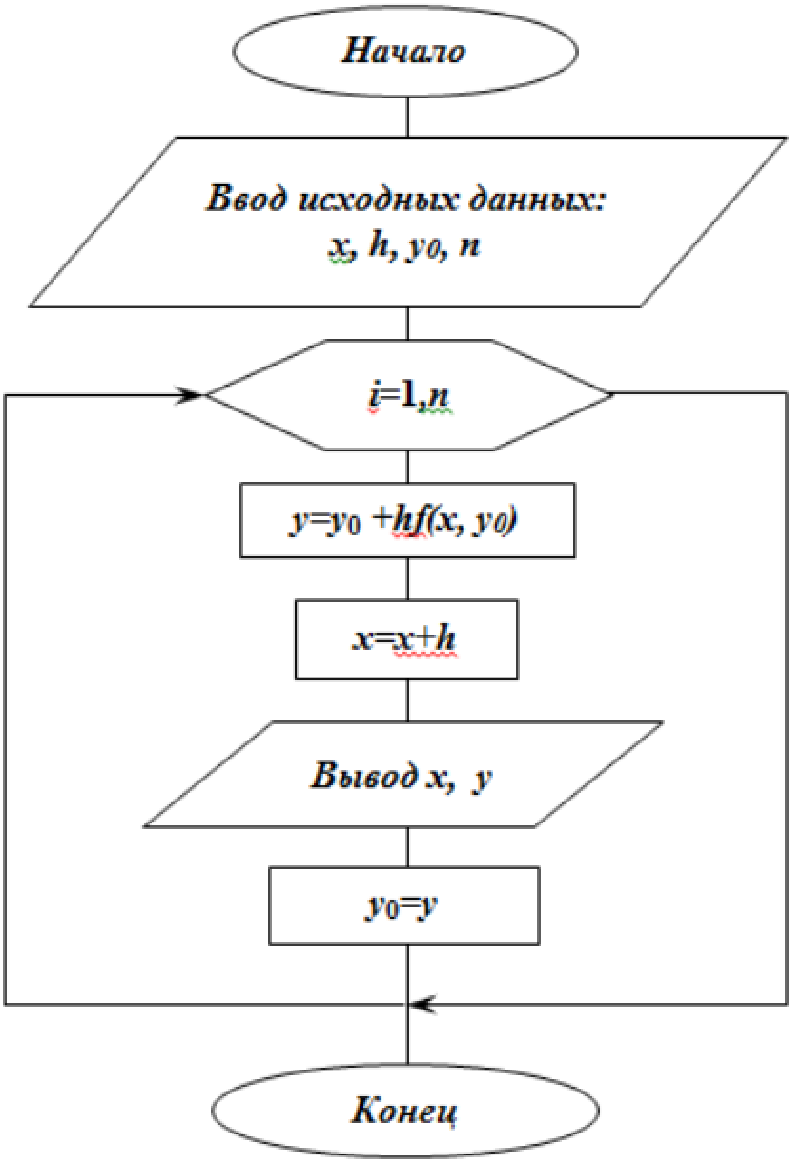


Рис.1 Блок-схема метода Эйлера

Широко распространен **метод Рунге-Кутта четвертого порядка**, часто без уточнений называемый просто методом Рунге – Кутты.

$$\boldsymbol{y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)}, \tag{13}$$

$$\boldsymbol{k_1 = h \cdot f(x_i, y_i)}$$

$$\boldsymbol{k_2 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_1}{2})}$$

$$\boldsymbol{k_3 = h \cdot f(x_i + \frac{h}{2}, y_i + \frac{k_2}{2})}$$

$$\boldsymbol{k_4 = h \cdot f(x_i + h, y_i + k_3)}$$

Таким образом, данный метод Рунге-Кутта требует на каждом шаге четырехкратного вычисления правой части $f(x, y)$уравнения (5).
Суммарная погрешность этого метода есть величина $\delta_n = O(h^4)$.

Метод Милна

Метод Милна относится к многошаговым методам и представляет один из методов прогноза и коррекции.

Для получения формул Милна используется первая интерполяционная формула Ньютона с разностями до третьего порядка.

Решение в следующей точке находится в два этапа. На первом этапе осуществляется прогноз значения функции, а затем на втором этапе - коррекция полученного значения. Если полученное значение y после коррекции существенно отличается от спрогнозированного, то проводят еще один этап коррекции. Если опять имеет место существенное отличие от предыдущего значения (т.е. от предыдущей коррекции), то проводят еще одну коррекцию и т.д.

Вычислительные формулы:

а) этап прогноза:

$$y_i^{\text{прогн}} = y_{i-4} + \frac{4h}{3}(2f_{i-3} - f_{i-2} + 2f_{i-1})$$

б) этап коррекции:

$$y_i^{\text{корр}} = y_{i-2} + \frac{h}{3}(f_{i-2} + 4f_{i-1} + f_i^{\text{прогн}})$$

$$f_i^{\text{прогн}} = f(x_i, y_i^{\text{прогн}})$$

Для начала счета требуется задать решения в трех первых точках, которые можно получить одношаговыми методами (например, методом Рунге-Кутта).

Метод требует несколько меньшего количества вычислений (достаточно только два раза вычислить $f(x, y)$, остальные берутся с предыдущих этапов).
Суммарная погрешность этого метода есть величина $\delta_n = O(h^4)$.

ОЦЕНКА ПОГРЕШНОСТИ ЧИСЛЕННОГО РЕШЕНИЯ

Оценить погрешность приближенных решений можно двумя способами двумя способами:

1. $\varepsilon = \max_{0\le i\le n}|y_{i\text{точн}} - y_i|$,

где $y_{i\text{точн}}, y_i$ - значения точного и приближенного решений в узлах сетки x_i, $i = 1, \dots, n$

2. по правилу Рунге:

$R = \frac{y^h - y^{h/2}}{2^p - 1}$,

где y^h - решение задачи Коши с шагом h в точке $x + h$

$y^{h/2}$ - решение задачи Коши с шагом $h/2$ в точке $x + h$

p – порядок точности метода.

При использовании правила Рунге **контроль точности можно осуществлять:**

1. **На каждом шаге *h***.

Для этого вычисляем значение y_1 сначала с шагом *h*, затем с шагом *h*/2.

Если $\frac{\left|y_1^h - y_1^{h/2}\right|}{2^p - 1} \le \varepsilon$, тогда переходим к следующему узлу сетки $x_2 = x_1 + h$.

Если правило Рунге не работает, уменьшаем шаг и производим вычисления y_1 с шагом *h*/4:

$\frac{\left|y_1^{h/2} - y_1^{h/4}\right|}{2^p - 1} \le \varepsilon$ и т.д.

2. **На конце заданного интервала**.

Для этого численно решаем задачу с заданным шагом *h* на всем заданном интервале, затем решаем задачу с шагом *h*/2 на всем заданном интервале.

Сравниваем $\frac{\left|y_n^h - y_n^{h/2}\right|}{2^p - 1} \le \varepsilon$. Если точность не достигнута, шаг уменьшаем.

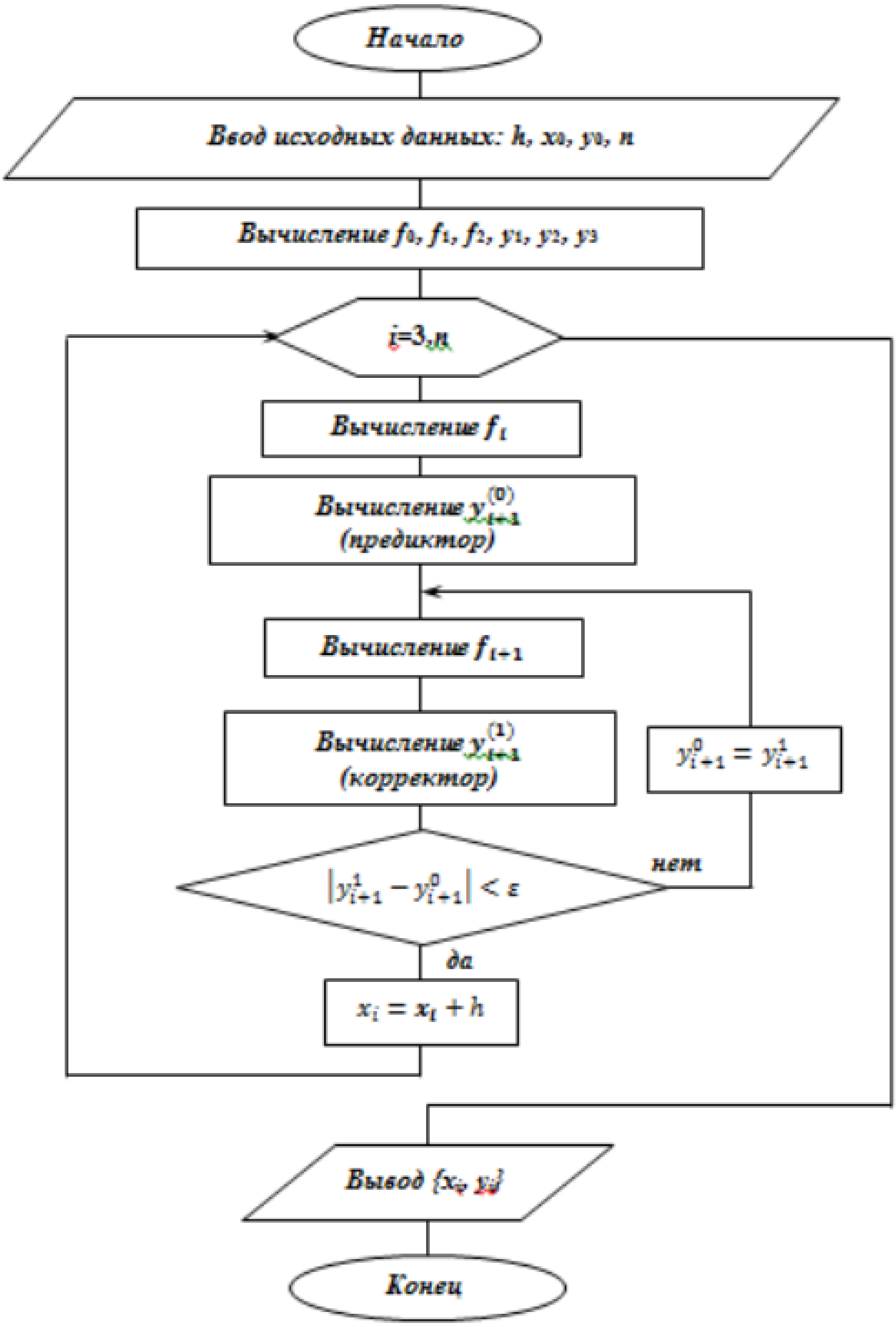


Рис. 3. Блок-схема метода предиктор-корректор

Программная реализация задачи

Листнинг программы:

```python
2 usages
def euler_method(func, y0, t0, t1, h, e, max_step_reductions=1000):
    t = t0
    y = np.array(y0)
    results = [(t, y)]
    step_reductions = 0

    table = PrettyTable()
    table.title = "Проверка по правилу Рунге"
    table.field_names = ["t", "y", "R (Runge)"]
    table.float_format = ".5"

    while t < t1:
        y_h = y + h * func(t, y)[0]
        y_half_h = y + (h / 2) * func(t, y)[0]
        print(t, 'при шаге h:', h, y_h, 'при шаге h/2:', h / 2, y_half_h)

        R = np.max(np.abs(y_half_h - y_h)) / (2 ** 1 - 1)

        table.add_row([f"{t:.5f}", f"{y_h[0]:.5f}", f"{R:.5f}"])

        if R <= e:
            y = y_h
            t = t + h
            results.append((t, y))
        else:
            h /= 2
            step_reductions += 1
            print("Шаг был уменьшен до", h)
            if step_reductions > max_step_reductions:
                print("Достигнуто максимальное количество уменьшений шага.")
                break

    print(table)
    return results
```

```python
def runge_kutta_method(func, y0, t0, t1, h, e, max_step_reductions=1000):
    t = t0
    y = np.array(y0)
    results = [(t, y[0])]
    step_reductions = 0

    table = PrettyTable()
    table.title = "Решение методом Рунге-Кутты"
    table.field_names = ["t", "y", "R"]
    table.float_format = ".5"

    while t < t1:
        k1 = h * func(t, y)[0]
        k2 = h * func(t + 0.5 * h, y + 0.5 * k1)[0]
        k3 = h * func(t + 0.5 * h, y + 0.5 * k2)[0]
        k4 = h * func(t + h, y + k3)[0]
        y_h = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6

        h /= 2
        k1 = h * func(t, y)[0]
        k2 = h * func(t + 0.5 * h, y + 0.5 * k1)[0]
        k3 = h * func(t + 0.5 * h, y + 0.5 * k2)[0]
        k4 = h * func(t + h, y + k3)[0]
        y_half_h = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6
        h *= 2

        R = np.max(np.abs(y_half_h - y_h)) / (2 ** 4 - 1)

        table.add_row([f"{t:.5f}", f"{y_h[0]:.5f}", f"{R:.5f}"])

        if R <= e:
            y = y_h
            t = t + h
            results.append((t, y[0]))
        else:
            h /= 2
            step_reductions += 1
            print("Шаг был уменьшен до", h)
            if step_reductions > max_step_reductions:
                print("Достигнуто максимальное количество уменьшений шага.")
                break

    print(table)
    return results
```

```python
def milne_method(func, y0, t0, t1, h, e, exact_solution, max_step_reductions=1000):
    results = runge_kutta_method(func, y0, t0, t1, h, e)

    if len(results) < 4:
        print("Ошибка: Не удалось получить первые три точки методом Рунге-Кутты.")
        return []

    t = t0
    x_needed = [t0, t0 + h, t0 + 2 * h, t0 + 3 * h]
    y_list = []
    counter = 0
    while counter != 4:
        for x, y in results:
            if abs(x - x_needed[counter]) < 0.0000001:
                y_list.append(y)
                counter += 1
                break

    while len(y_list) < ((t1 - t0) / h):
        y_list.append(y_list[0])
```

```python
        R = np.abs(exact_solution(t + (i + 1) * h) - y_corr)
        if R > e:
            h /= 2
            step_reductions += 1
            print("Шаг был уменьшен до", h)
            if step_reductions > max_step_reductions:
                print("Достигнуто максимальное количество уменьшений шага.")
                break
            i -= 1
        else:
            table.add_row([f"{t + (i + 1) * h:.5f}", f"{y_corr:.5f}", f"{R:.5f}"])
            i += 1

    print(table)

    x = np.arange(t0, t1 + h, h)
    results = ([(xi, yi) for xi, yi in zip(x, y_list)])
    return results
```

```python
    table = PrettyTable()
    table.title = "Метод Милна"
    table.field_names = ["t", "y", "R"]
    table.float_format = ".5"

    i = 3
    step_reductions = 0

    while i < (len(y_list) - 1):
        y_pred = y_list[i - 3] + 4 * h * (
                2 * func(t + (i - 2) * h, y_list[i - 2]) - func(t + (i - 1) * h, y_list[i - 1]) +
                2 * func(t + i * h, y_list[i])) / 3

        y_corr = y_list[i - 1] + h * (
                func(t + (i - 1) * h, y_list[i - 1]) + 4 * func(t + i * h, y_list[i]) +
                func(t + (i + 1) * h, y_pred)) / 3
        while np.abs(y_corr - y_pred) > e:
            y_pred = y_corr
            y_corr = y_list[i - 1] + h * (
                    func(t + (i - 1) * h, y_list[i - 1]) + 4 * func(t + i * h, y_list[i]) +
                    func(t + (i + 1) * h, y_pred)) / 3

        y_list[i + 1] = y_corr
```

Результат работы программы:

```
Выберите систему уравнений:
1. y' = y + (1 + x) * y^2
2. y' = sin(x) - y
3. y' = -y + x^2
Введите номер системы: 1

Выберите метод решения:
1. Метод Эйлера
2. Метод Рунге-Кутты 4-го порядка
3. Метод Милна
Введите номер метода: 3
Начальное значение x: 1
Начальное значение y: -1
[0.0]
Шаг h: 0.5
Количество шагов n: 10
Введите требуемую точность e: 0.01
```

```
+-----------------------------+
| Решение методом Рунге-Кутты |
+---------+----------+---------+
|    t    |    y     |    R    |
+---------+----------+---------+
| 1.00000 | -0.66931 | 0.00872 |
| 1.50000 | -0.50172 | 0.00475 |
| 2.00000 | -0.40097 | 0.00297 |
| 2.50000 | -0.33387 | 0.00203 |
| 3.00000 | -0.28602 | 0.00147 |
| 3.50000 | -0.25018 | 0.00111 |
| 4.00000 | -0.22233 | 0.00087 |
| 4.50000 | -0.20007 | 0.00070 |
| 5.00000 | -0.18186 | 0.00058 |
| 5.50000 | -0.16670 | 0.00048 |
+---------+----------+---------+
```

Метод Милна

t	y	R
3.00000	-0.33471	0.00138
3.50000	-0.28420	0.00152
4.00000	-0.25102	0.00102
4.50000	-0.22125	0.00097
5.00000	-0.19983	0.00017
5.50000	-0.18214	0.00032

Решение

i	xi	yi	f(xi, yi)	Точное решение
0	1.00000	-1.00000	1.00000	-1.00000000000000
1	1.50000	-0.66931	0.45063	-0.666666666666667
2	2.00000	-0.50172	0.25346	-0.500000000000000
3	2.50000	-0.40097	0.16175	-0.400000000000000
4	3.00000	-0.33471	0.11342	-0.333333333333333
5	3.50000	-0.28420	0.07926	-0.285714285714286
6	4.00000	-0.25102	0.06404	-0.250000000000000
7	4.50000	-0.22125	0.04798	-0.222222222222222
8	5.00000	-0.19983	0.03976	-0.200000000000000
9	5.50000	-0.18214	0.03349	-0.181818181818182

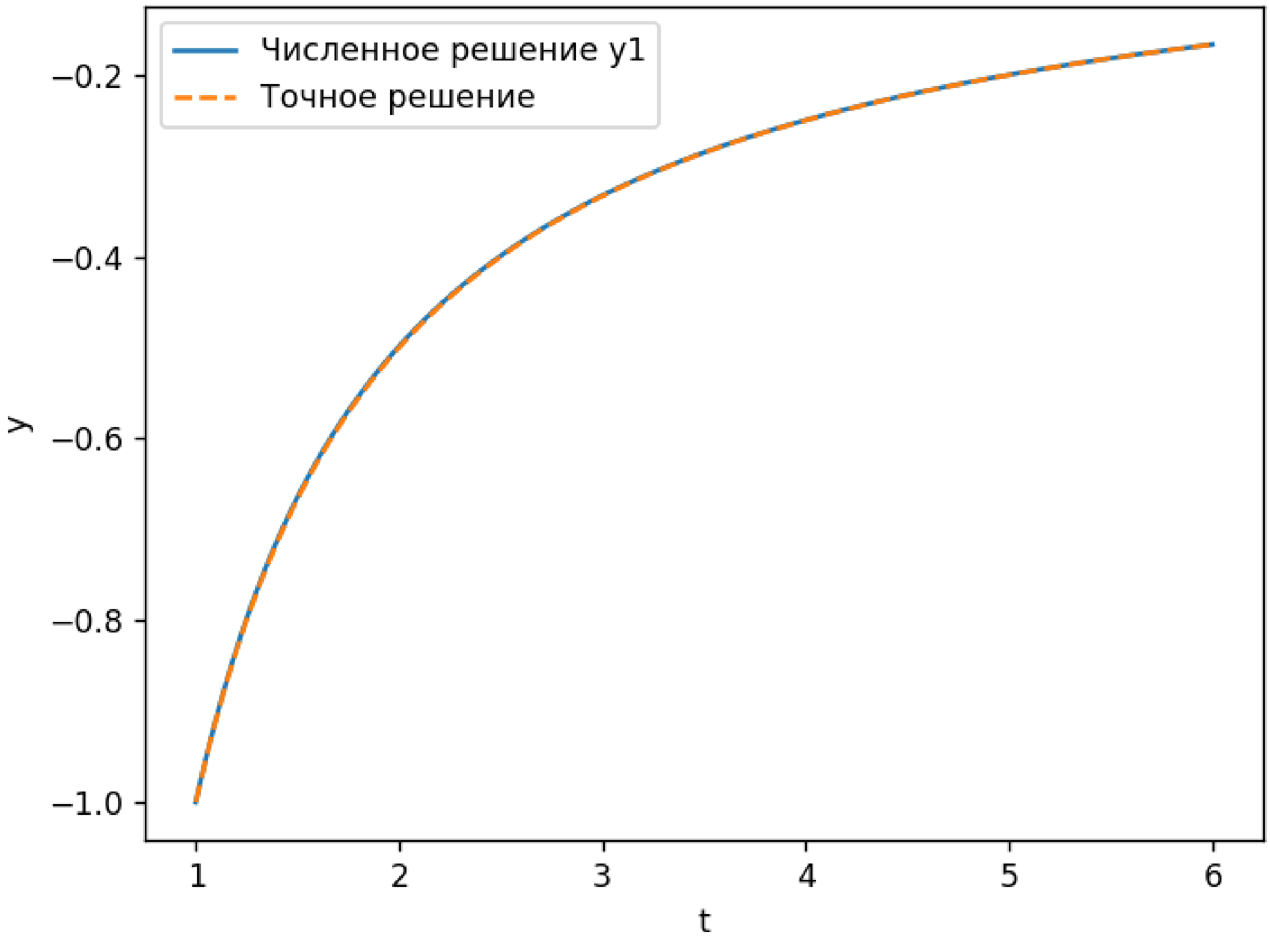
```
Выберите систему уравнений:
1. y' = y + (1 + x) * y^2
2. y' = sin(x) - y
3. y' = -y + x^2
Введите номер системы: 1

Выберите метод решения:
1. Метод Эйлера
2. Метод Рунге-Кутты 4-го порядка
3. Метод Милна
Введите номер метода: 1
Начальное значение x: 1
Начальное значение y: -1
[0.0]
Шаг h: 0.5
Количество шагов n: 10
Введите требуемую точность e: 0.01
Шаг был уменьшен до 0.25
Шаг был уменьшен до 0.125
Шаг был уменьшен до 0.0625
Шаг был уменьшен до 0.03125
Шаг был уменьшен до 0.015625
```

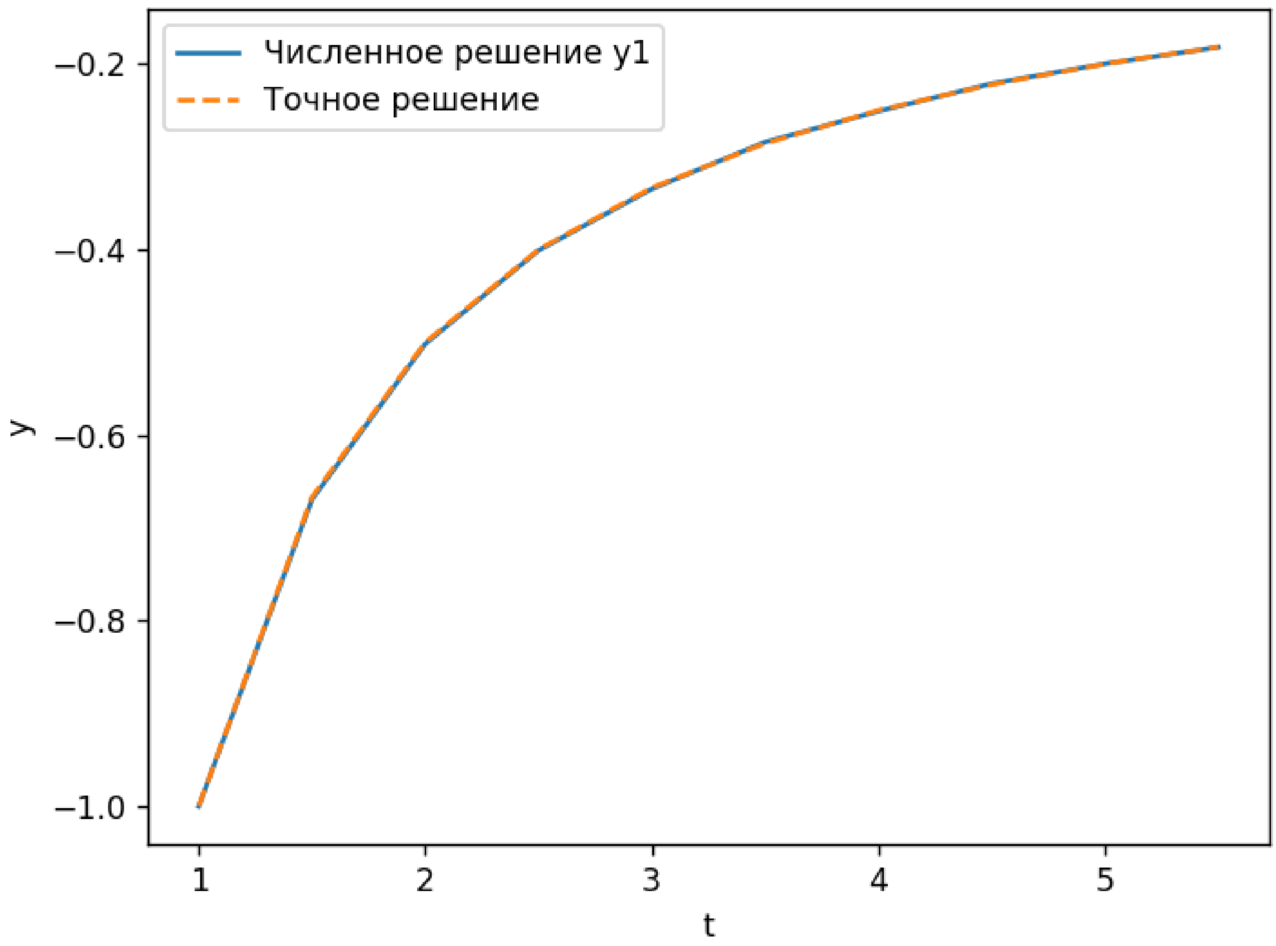
Проверка по правилу Рунге		
t	y	R (Runge)
1.00000	-0.50000	0.25000
1.00000	-0.75000	0.12500
1.00000	-0.87500	0.06250
1.00000	-0.93750	0.03125
1.00000	-0.96875	0.01562
1.00000	-0.98438	0.00781
1.01562	-0.96924	0.00757
1.03125	-0.95457	0.00734
1.04688	-0.94034	0.00711
1.06250	-0.92654	0.00690
1.07812	-0.91314	0.00670
1.09375	-0.90013	0.00651
1.10938	-0.88749	0.00632
1.12500	-0.87520	0.00614
1.14062	-0.86326	0.00597
1.15625	-0.85164	0.00581
1.17188	-0.84033	0.00565
1.18750	-0.82933	0.00550
1.20312	-0.81861	0.00536
1.21875	-0.80817	0.00522

(Еще очень много строчек…)

```
| 5.98438 | -0.16658 |  0.00022  |
+---------+---------+-----------+
+--------------------------------------------------------------+
|                           Решение                            |
+-----+---------+---------+----------+------------------+
|  i  |    xi   |    yi   | f(xi, yi) |   Точное решение   |
+-----+---------+---------+----------+------------------+
|  0  | 1.00000 | -1.00000 |  1.00000  | -1.00000000000000  |
|  1  | 1.01562 | -0.98438 |  0.96875  | -0.984615384615385 |
|  2  | 1.03125 | -0.96924 |  0.93896  | -0.969696969696970 |
|  3  | 1.04688 | -0.95457 |  0.91054  | -0.955223880597015 |
|  4  | 1.06250 | -0.94034 |  0.88340  | -0.941176470588235 |
|  5  | 1.07812 | -0.92654 |  0.85747  | -0.927536231884058 |
|  6  | 1.09375 | -0.91314 |  0.83268  | -0.914285714285714 |
|  7  | 1.10938 | -0.90013 |  0.80895  | -0.901408450704225 |
|  8  | 1.12500 | -0.88749 |  0.78624  | -0.888888888888889 |
|  9  | 1.14062 | -0.87520 |  0.76447  | -0.876712328767123 |
|  10 | 1.15625 | -0.86326 |  0.74361  | -0.864864864864865 |
|  11 | 1.17188 | -0.85164 |  0.72360  | -0.853333333333333 |
|  12 | 1.18750 | -0.84033 |  0.70439  | -0.842105263157895 |
|  13 | 1.20312 | -0.82933 |  0.68595  | -0.831168831168831 |
|  14 | 1.21875 | -0.81861 |  0.66822  | -0.820512820512820 |
|  15 | 1.23438 | -0.80817 |  0.65118  | -0.810126582278481 |
|  16 | 1.25000 | -0.79799 |  0.63479  | -0.800000000000000 |
|  17 | 1.26562 | -0.78807 |  0.61902  | -0.790123456790124 |
|  18 | 1.28125 | -0.77840 |  0.60383  | -0.780487804878049 |
|  19 | 1.29688 | -0.76897 |  0.58920  | -0.771084337349398 |
```
…

303	5.73438	-0.17428	0.03027	-0.174386920980926
304	5.75000	-0.17381	0.03011	-0.173913043478261
305	5.76562	-0.17334	0.02994	-0.173441734417344
306	5.78125	-0.17287	0.02978	-0.172972972972973
307	5.79688	-0.17241	0.02962	-0.172506738544474
308	5.81250	-0.17194	0.02947	-0.172043010752688
309	5.82812	-0.17148	0.02931	-0.171581769436997
310	5.84375	-0.17103	0.02915	-0.171122994652406
311	5.85938	-0.17057	0.02900	-0.170666666666667
312	5.87500	-0.17012	0.02884	-0.170212765957447
313	5.89062	-0.16967	0.02869	-0.169761273209549
314	5.90625	-0.16922	0.02854	-0.169312169312169
315	5.92188	-0.16877	0.02839	-0.168865435356201
316	5.93750	-0.16833	0.02824	-0.168421052631579
317	5.95312	-0.16789	0.02809	-0.167979002624672
318	5.96875	-0.16745	0.02795	-0.167539267015707
319	5.98438	-0.16701	0.02780	-0.167101827676240
320	6.00000	-0.16658	0.02766	-0.166666666666667


Численное решение y1
Точное решение
y
t
−0.2
−0.4
−0.6
−0.8
−1.0
1
2
3
4
5
6



Исходный код: https://github.com/tsostanov/CompMath6

Вывод

В ходе работы были изучены различные методы решения дифференциальных уравнений, были программно реализованы методы Ньютона, Рунге-Кутта и Милна