МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В.Ломоносова

Факультет вычислительной математики и кибернетики

Компьютерный практикум по учебному курсу «ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»

ЗАДАНИЕ № 2.

Численные методы решения дифференциальных уравнений

ОТЧЕТ

о выполненном задании

студента 204 учебной группы факультета ВМК МГУ

Скрябина Глеба Денисовича

(фамилия, имя, отчество)

гор. Москва

2020 г.

Оглавление

Подвариант 1

Постановка задачи и ее цели

Целью работы является освоение метода Рунге-Кутта второго и четвертого порядка точности, применяемые для численного решения задачи Коши для дифференциального уравнения и системы дифференциальных уравнений первого порядка.

Постановка задачи.

Рассматривается обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной и имеющее вид:

$$\frac{dy}{dx} = f(x, y), \ x_0 < x, \tag{1}$$

с дополнительным начальным условием, заданным в точке $x = x_0$:

$$y(x_0) = y_0. \tag{2}$$

Предполагается, что правая часть уравнения (1) функция $f = f(x, y)$ такова, что гарантирует существование и единственность решения задачи Коши (1)-(2).

В том случае, если рассматривается не одно дифференциальное уравнение вида (1), а система обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производных неизвестных функций, то соответствующая задача Коши имеет вид (на примере двух дифференциальных уравнений):

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2), \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2), \ x > x_0. \end{cases} \tag{3}$$

Дополнительные (начальные) условия задаются в точке $x = x_0$:

$$y_1(x_0) = y_1^{(0)}, \ y_2(x_0) = y_2^{(0)}. \tag{4}$$

Также предполагается, что правые части уравнений из (3) заданы так, что это гарантирует существование и единственность решения задачи Коши (3)-(4), но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешенной относительно производных неизвестных функций.

Необходимо:

1. Решить задачу Коши (1)-(2) (или (3)-(4)) наиболее известными и широко используемыми на практике методами Рунге-Кутта второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой (на равномерной сетке); полученное конечно-разностное уравнение (или уравнения в случае системы), представляющее фактически некоторую рекуррентную формулу, просчитать численно;
2. Найти численное решение задачи и построить его график;
3. Найденное численное решение сравнить с точным решением дифференциального уравнения (подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при проверке можно использовать ресурсы on-line системы http://www.wolframalpha.com или пакета Maple и т.п.).

Задание 1.

$$f(x,y) = -y - x^2,$$
$$(x_0, y_0) = (0,10),$$

Точное решение $y = y(x) = -x^2 + 2x - 2 + 12e^{-x}$.

Задание 2.

$$f_1(x,u,v) = \sin(1.4 \cdot u^2) - x + v,$$
$$f_2(x,u,v) = x + u - 2.2 \cdot v^2 + 1,$$
$$x_0 = 0,\ y_1^{(0)} = 1,\ y_2^{(0)} = 0.5$$

Метод решения

Метод Рунге-Кутта второго порядка точности

Рассмотрим задачу Коши для дифференциального уравнения первого порядка

$$u' = f(x,u),$$

$$u(x_0) = u_0$$

на отрезке [*x*0, *x*0 +*l*] длины *l*. Возьмем некоторое целое число *n*, введем шаг *h* = *l* / *n* и образуем на этом отрезке сетку

$$x_i = x_0 + ih,\ 0 \le i \le n.$$

Метод Рунге-Кутта второго порядка точности представляет собой рекуррентное соотношение следующего вида:

$$y_{i+1} = y_i + \left[(1-\alpha) f(x_i, y_i) + \alpha f\left(x_i + \frac{h}{2\alpha}, y_i + \frac{h}{2\alpha} f(x_i, y_i)\right)\right] h.$$

Наиболее удобные разностные схемы этого семейства соответствуют двум значениям параметра α: α = 1/2 и α =1. При α = 1/2 рекуррентная формула (62) принимает вид

$$y_{i+1} = y_i + \frac{h}{2}\left\{f(x_i, y_i) + f\left(x_i + h, y_i + hf(x_i, y_i)\right)\right\}$$

Метод Рунге-Кутта четвертого порядка точности

На практике обычно используется более точная схема Рунге-Кутта четвертого порядка точности, которая имеет следующий вид:

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

где

$$k_1 = f(x_i, y_i), \quad k_2 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_1\right),$$
$$k_3 = f\left(x_i + \frac{h}{2}, y_i + \frac{h}{2}k_2\right), \quad k_4 = f(x_i + h, y_i + hk_3)$$

Метод Рунге-Кутта четвертого порядка точности для систем

Рассмотрим задачу Коши для системы обыкновенных дифференциальных уравнений

$$\left\{\begin{array}{lcl} y_1' & = & f_1(x, y_1, \ldots, y_n) \\ & \ldots & \\ y_n' & = & f_n(x, y_1, \ldots, y_n) \\ y_1(x_0) & = & y_{01} \\ & \ldots & \\ y_n(x_0) & = & y_{0n} \end{array}\right\} \iff \left\{\begin{array}{lcl} \mathbf{y}' & = & \mathbf{f}(x, \mathbf{y}) \\ \mathbf{y}(x_0) & = & \mathbf{y_0} \end{array}\right.$$

Приближенное значение в последующих точках по методу Рунге-Кутта четвертого порядка вычисляется по итерационной формуле:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

При этом вычисление нового значения проходит в четыре стадии:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(x_n, \mathbf{y}_n), \\ \mathbf{k}_2 &= \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right), \\ \mathbf{k}_3 &= \mathbf{f}\left(x_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right), \\ \mathbf{k}_4 &= \mathbf{f}(x_n + h, \mathbf{y}_n + h\,\mathbf{k}_3). \end{aligned}$$

Описание программы

Методы Рунге-Кутта 2 и 4 порядков:

```python
# Метод Рунге-Кутта 2 порядка точности:
def rk2(f, x0, y0, l, n):
    h = l / n
    # делим отрезок [x0, x0 + l] на n частей:
    X = [x0 + i * h for i in range(n + 1)]
    Y = [y0] # y(x0) = y0
    for i in range(n):
        xi = X[i]
        yi = Y[i]
        # Вычисление вспомогательных коэффициентов:
        k1 = f(xi, yi)
        k2 = f(xi + h, yi + k1 * h)
        # По рекурентной формуле вычисляем следующий y:
        Y.append(yi + h * (k1 + k2) / 2)
    return X, Y
```

```python
# Метод Рунге-Кутта 4 порядка точности:
def rk4(f, x0, y0, l, n):
    h = l / n
    # делим отрезок [x0, x0 + l] на n частей:
    X = [x0 + i * h for i in range(n + 1)]
    Y = [y0] # y(x0) = y0
    for i in range(n):
        xi = X[i]
        yi = Y[i]
        # Вычисление вспомогательных коэффициентов:
        k1 = f(xi, yi)
        k2 = f(xi + h / 2, yi + k1 * h / 2)
        k3 = f(xi + h / 2, yi + k2 * h / 2)
        k4 = f(xi + h, yi + k3 * h)
        # По рекурентной формуле вычисляем следующий y:
        Y.append(yi + h * (k1 + 2 * k2 + 2 * k3 + k4) / 6)
    return X, Y
```

Метод Рунге-Кутта для системы:

```
# Метод Рунге-Кутта 4 порядка точности для системы:
def rk4_sys(fs, x0, y0, l, n):
    h = l / n
    # делим отрезок [x0, x0 + l] на n частей:
    X = [x0 + i * h for i in range(n + 1)]
    Y = [y0] # y(x0) = y0
    indexGen = range(len(y0))
    f = lambda x, y: [fs[0](x, y), fs[1](x, y)]
    for i in range(n):
        xi = X[i]
        yi = Y[i]
        # Вычисление вспомогательных коэффициентов:
        k1 = f(xi, yi)
        tmp = [yi[j] + k1[j] * h / 2 for j in indexGen]
        k2 = f(xi + h / 2, tmp)
        tmp = [yi[j] + k2[j] * h / 2 for j in indexGen]
        k3 = f(xi + h / 2, tmp)
        tmp = [yi[j] + k3[j] * h for j in indexGen]
        k4 = f(xi + h, tmp)
        # По рекурентной формуле вычисляем следующий y:
        tmp = [k1[j] + 2 * (k3[j] + k3[j]) + k4[j] for j in indexGen]
        Y.append([yi[j] + tmp[j] * h / 6 for j in indexGen])
    return X, Y
```

Решения Задач Коши и построения графиков:

```
# Задача Коши
def CauchyProblem(x0, y0, l, f, yreal):
    # Решаем задачу Коши методом Рунге-Кутта 2 и 4 порядка точности:
    ns = [5, 10, 25]
    for MethodIndex, rk in zip([2, 4], [rk2, rk4]):
        fig = plt.figure()
        X, Y = getReal(yreal, x0, l)
        plt.plot(X, Y, marker = 'o', color='greenyellow', label='Точное решение')
        for n, c in zip(ns, COLORS):
            X, Y = rk(f, x0, y0, l, n)
            plt.plot(X, Y, color=c, label=f'{n} шагов')

        # Оформляем графики:
        plt.gcf().canvas.set_window_title('Подвариант 1. Для продолжения закройте это окно')
        plt.title(f'Метод Рунге-Кутта {MethodIndex}-го порядка точности')
        plt.legend(loc='lower left')
        fig.set_figheight(WINDOW_HEIGHT)
        fig.set_figwidth(WINDOW_WIDTH)
        plt.grid()
        plt.show()
```

```python
# Задача Коши для системы
def CauchyProblemForSys(x0, y0, l, fs):
    # Решаем задачу Коши для Системы методом Рунге-Кутта 4 порядка точности:
    ns = [10, 15, 45]
    fig = plt.figure()
    ax = fig.add_subplot(projection='3d')
    for n, c in zip(ns, COLORS):
        X, Y = rk4_sys(fs, x0, y0, l, n)
        Ydivided = [[y[i] for y in Y] for i in [0, 1]]
        ax.plot(X, Ydivided[0], Ydivided[1], color=c, label=f'{n} шагов')

    # Оформляем графики:
    plt.gcf().canvas.set_window_title('Подвариант 1. Для продолжения закройте это окно')
    plt.title('Метод Рунге-Кутта 4-го порядка точности для системы\nГрафик можно крутить курсором')
    fig.set_figheight(WINDOW_HEIGHT)
    fig.set_figwidth(WINDOW_WIDTH)
    ax.set_xlabel('X')
    ax.set_ylabel('U')
    ax.set_zlabel('V')
    plt.legend()
    plt.grid()
    plt.show()
```

Входные данные и тесты:

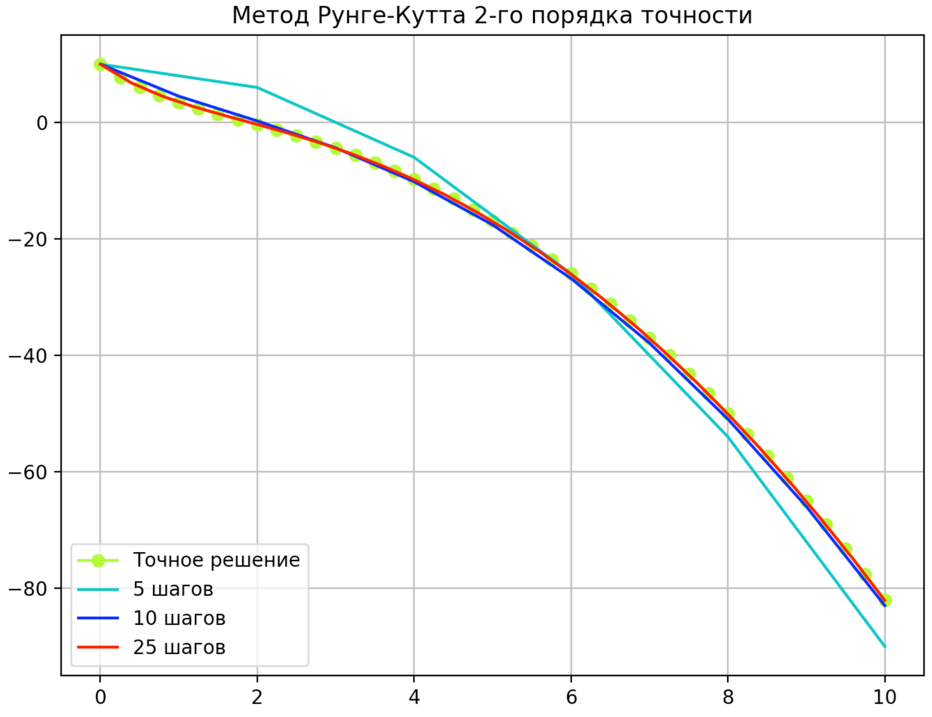
```python
def main():
    # Вариант 1 - 3
    # Определяем начальные данные:
    x0 = 0
    y0 = 10
    l = 10
    f = lambda x, y: -y - x ** 2
    yreal = lambda x: -x ** 2 + 2 * x - 2 + 12 * exp(-x)
    CauchyProblem(x0, y0, l, f, yreal)
    # Тест для проверки метода:
    x0 = -1
    y0 = -3
    l = 2
    f = lambda x, y:  2*x*y + y*x**2
    yreal = lambda x: -3 * exp((x ** 3) / 3 + (x ** 2) - 2 / 3)
    CauchyProblem(x0, y0, l, f, yreal)

    # Вариант 2 - 17
    # Определяем начальные данные:
    x0 = 0
    y0 = [1, 0.5]
    l = 6
    f1 = lambda x, y: sin(1.4 * y[0] ** 2) - x + y[1]
    f2 = lambda x, y: x + y[0] - 2.2 * y[1] ** 2 + 1
    fs = [f1, f2]
    CauchyProblemForSys(x0, y0, l, fs)
```

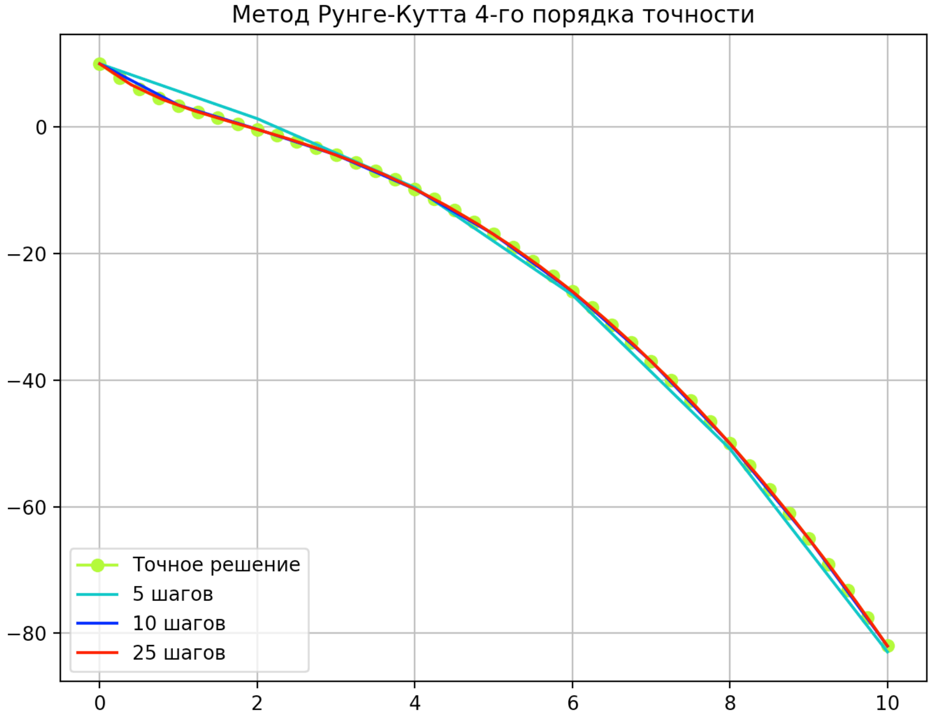
Тестирование программы

$f(x, y) = -y - x^2$

$\{x_0 = 0,\ y_0 = 10\}$

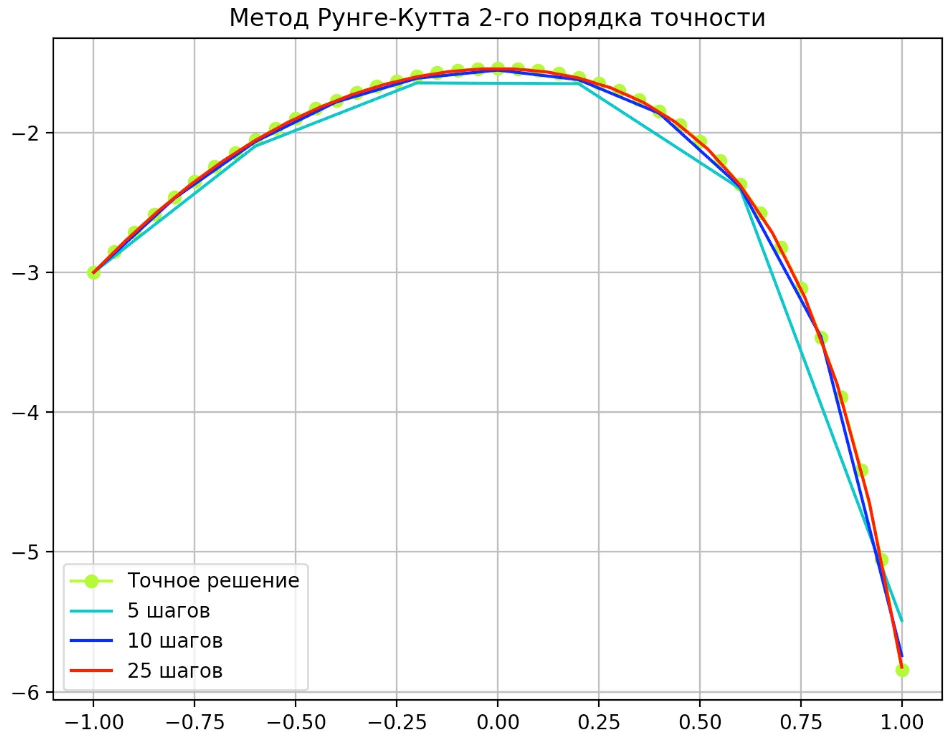
Точное решение $y = y(x) = -x^2 + 2x - 2 + 12e^{-x}$

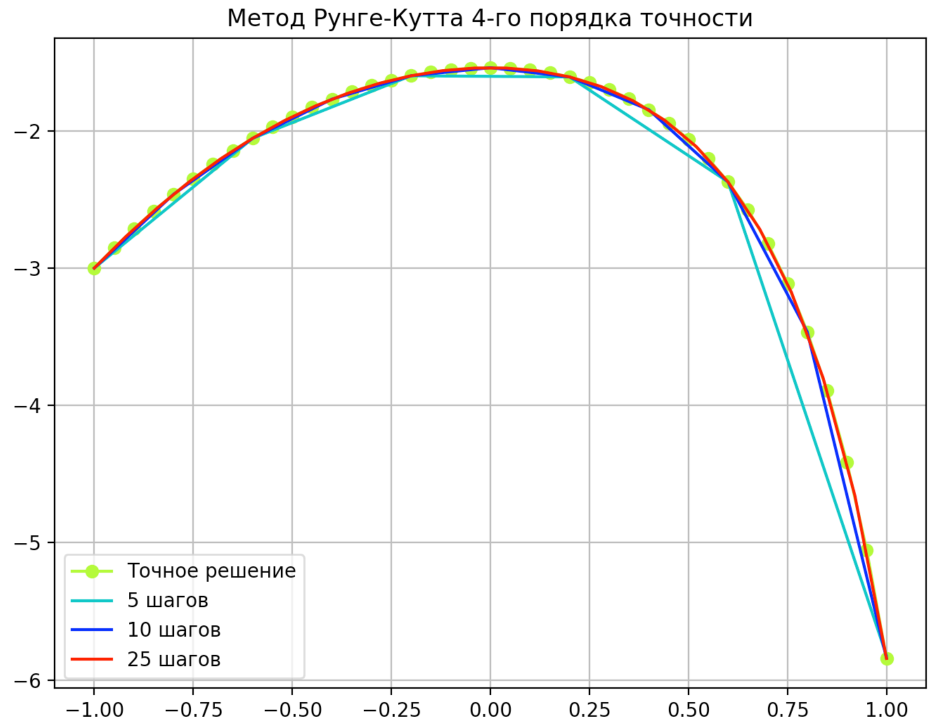




$$f(x, y) = 2\,x\,y + y\,x^2$$
$$\{x_0 = -1,\ y_0 = -3\}$$

Точное решение $y = y(x) = -3\,e^{x^3/3 + x^2 - 2/3}$

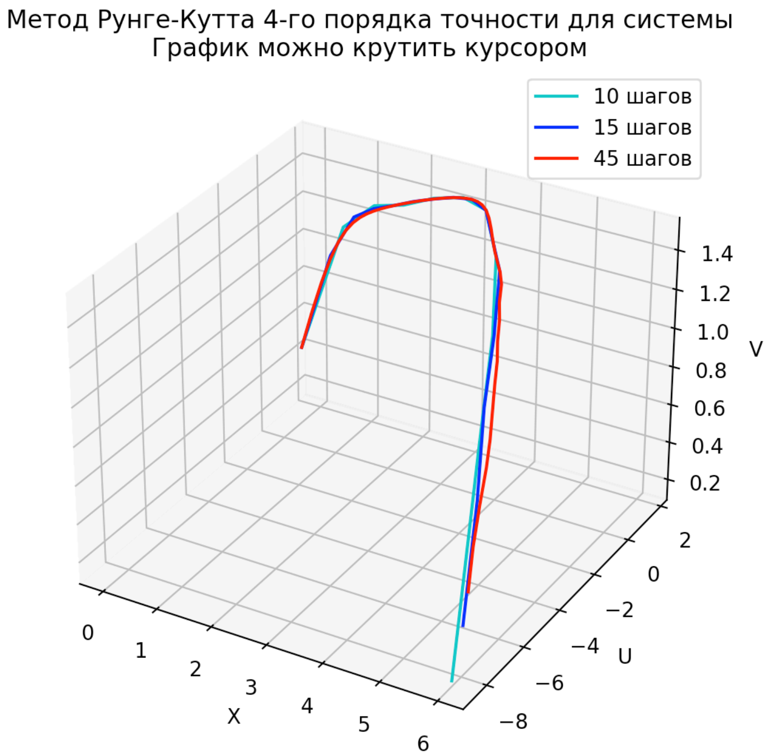


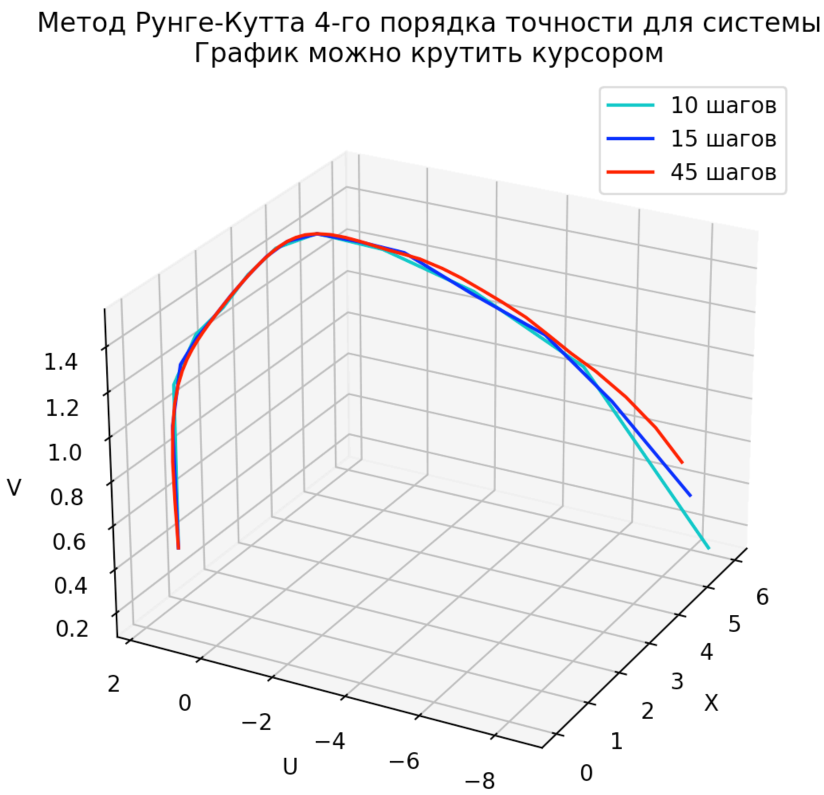


$f1(x, y) = \sin(1.4 \cdot u^2) - x + v$

$f2(x, y) = x + u - 2.2 \cdot v^2 + 1$

$x_0 = 0, \ y_1^{(0)} = u\_0 = 1, \ y_2^{(0)} = v\_0 = 0.5$





Выводы

В ходе выполнения поставленной задачи были реализованы методы Рунге-Кутта второго и четвертого порядка точности для решения задачи Коши. Они сложнее метода Эйлера, но вместе с этим точнее. На практике обычно используют метод Рунге-Кутта 4-го порядка.

Подвариант 2

Постановка задачи и ее цели

Целью работы является освоение метода прогонки решения краевой задачи для дифференциального уравнения второго порядка.

Постановка задачи.

Рассматривается линейное дифференциальное уравнение второго порядка вида

$$y'' + p(x)\cdot y' + q(x)\cdot y = -f(x),\ 1 < x < 0, \tag{1}$$

с дополнительными условиями в граничных точках

$$\begin{cases} \sigma_1 y(0) + \gamma_1 y'(0) = \delta_1, \\ \sigma_2 y(1) + \gamma_2 y'(1) = \delta_2. \end{cases} \tag{2}$$

Необходимо:

1) Решить краевую задачу (1)-(2) методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки;
2) Найти разностное решение задачи и построить его график;
3) Найденное разностное решение сравнить с точным решением дифференциального уравнения (подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при проверке можно использовать ресурсы on-line системы http://www.wolframalpha.com или пакета Maple и т.п.).

Задание:

$$y'' + 3y' - \frac{y}{x} = x + 1$$

$$y'(1.2) = 1$$

$$2y(1.5) - y'(1.5) = 0.5$$

Метод решения

Рассмотрим следующую задачу для линейного дифференциального уравнения второго порядка на отрезке $[a,b]$:

$$u'' - q(x)u = -f(x), \quad a < x < b,$$
$$u(a) = u_1, \quad u(b) = u_2$$

Возьмем некоторое целое число n, введем шаг $h=(b-a)/n$ и построим сетку

$$x_i = a + ih, \quad 0 \le i \le n$$

Заменим дифференциальное уравнение его разностным аналогом. В результате получим следующую задачу:

$$\frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} - q_i y_i = -f_i, \quad 1 \le i \le n-1,$$

$$y_0 = u_0, \quad y_n = u_2$$

Здесь $q_i = q(x_i), f_i = f(x_i)$, граничные условия для сеточной функции $\{y_i\}$ взяты такими же, что и в дифференциальной задаче. Разностные уравнения (107) можно переписать в виде

$$y_{i-1} - \left(2 + q_i h^2\right) y_i + y_{i+1} = -f_i h^2, \quad 1 \le i \le n-1.$$

Мы получили линейную систему из $(n-1)$-го уравнения с $(n-1)$-им неизвестным y_i, $1 \le i \le n-1$. Значения y_0 и y_n задаются граничными условиями.

Из записи разностных уравнений видно, что мы получили систему уравнений с трехдиагональной матрицей с диагональным преобладанием: диагональный элемент $(2 + qi\ h^2)$ больше суммы двух других элементов той же строки, равной 2. Диагональное преобладание гарантирует существование и единственность решения системы, которое может быть построено методом прогонки.

Метод прогонки.

Рассмотрим систему линейных уравнений вида

$$A_i x_{i-1} + C_i x_i + B_i x_{i+1} = F_i, \ i = 1, \ldots, n-1,$$

$$x_0 = q_0, \ x_n = q_n,$$

Пользуясь тем, что значения x_0 и x_n заданы, перепишем эту систему в виде:

$$\begin{aligned} &C_1 x_1 + B_1 x_2 && = F_1 - A_1 q_0 \\ &A_2 x_1 + C_2 x_2 + B_1 x_3 && = F_2 \\ &\vdots \\ &A_{n-1} x_{n-2} + C_{n-1} x_{n-1} && = F_{n-1} - B_{n-1} q_n \end{aligned}$$

Матрица этой системы имеет трёхдиагональную структуру:

$$\begin{bmatrix} C_1 & B_1 & 0 & 0 & \cdots & 0 & 0 \\ A_2 & C_2 & B_2 & 0 & \cdots & 0 & 0 \\ 0 & A_3 & C_3 & B_3 & \cdots & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & A_{n-1} & C_{n-1} \end{bmatrix}$$

Метод основан на предположении, что искомые неизвестные x_i и x_{i+1} связаны рекуррентным соотношением

$$x_i = \alpha_{i+1} x_{i+1} + \beta_{i+1},\ 0 \le i \le n-1$$

Здесь величины α_{i+1}, β_{i+1}, получившие название прогоночных коэффициентов, подлежат определению, исходя из условий задачи. Фактически такая процедура означает замену прямого определения неизвестных x_i задачей определения прогоночных коэффициентов с последующим расчетом по ним величин x_i.

Рекуррентные соотношения для прогоночных коэффициентов:

$$\alpha_{i+1} = \frac{-B_i}{A_i \alpha_i + C_i},\ \beta_{i+1} = \frac{F_i - A_i \beta_i}{A_i \alpha_i + C_i},\ i = 1, 2, \ldots n-1$$

$$\alpha_1 = 0, \quad \beta_1 = q_0$$

Описание программы

Класс для входных данных:

```python
class DifferentialEquation:
    def __init__(self, p, q, f, sigma, gamma, delta):
        # y''(x) + p(x) * y'(x) + q(x) * y(x) = f(x)
        self.p = p
        self.q = q
        self.f = f
        self.s = sigma
        self.g = gamma
        self.d = delta
```

Метод конечных разностей:

```python
def finiteDifferenceMethod(de, x0, xn, n):
    h = (xn - x0) / n
    # делим отрезок [x0, x0 + l] на n частей:
    X = [x0 + i * h for i in range(n + 1)]
    # Определяем вспомогательные коэффициенты и функции:
    A = lambda i: h ** -2 - de.p(X[i]) / h / 2
    B = lambda i: h ** -2 + de.p(X[i]) / h / 2
    C = lambda i: -(2 / (h ** 2)) + de.q(X[i])
    F = lambda i: de.f(X[i])

    a = [0, -de.g[0] / (h * de.s[0] - de.g[0])]
    b = [0, de.d[0] / (de.s[0] - de.g[0] / h)]
    # Вычисление вспомогательных коэффициентов:
    for i in range(1, n):
        tmpdiv = A(i) * a[i] + C(i)
        a.append(-B(i) / tmpdiv)
        b.append((F(i) - A(i) * b[i]) / tmpdiv)
    m = [[1, -a[n]],
         [1, -(h * de.s[1] + de.g[1]) / de.g[1]]]
    t = [b[n], -h * de.d[1] / de.g[1]]
    matr_ans = nplMatrSolve(m, t)
    # Начинаем расчет Y:
    Y = [None for i in range(n + 1)]
    Y[n], Y[n - 1] = matr_ans[1], matr_ans[0]
    # По рекурентной формуле вычисляем следующий y:
    for i in range(n - 2, -1, -1):
        Y[i] = a[i + 1] * Y[i + 1] + b[i + 1]

    return X, Y
```

Решение краевой задачи:

```python
def boundaryValueProblem(x0, xn, diffEq, ns, yreal=None):
    # Решаем Краевую задачу
    fig = plt.figure()
    if yreal != None:
        X, Y = getReal(yreal, x0, xn)
        plt.plot(X, Y, marker = 'o', color='greenyellow', label='Точное решение')
    for n, c in zip(ns, COLORS):
        X, Y = finiteDifferenceMethod(diffEq, x0, xn, n)
        plt.plot(X, Y, color=c, label=f'{n} шагов')

    # Оформляем графики:
    plt.gcf().canvas.set_window_title('Подвариант 2. Для продолжения закройте это окно')
    plt.title('Метод конечных разностей')
    fig.set_figheight(WINDOW_HEIGHT)
    fig.set_figwidth(WINDOW_WIDTH)
    plt.legend()
    plt.grid()
    plt.show()
```

Входные данные и тесты:

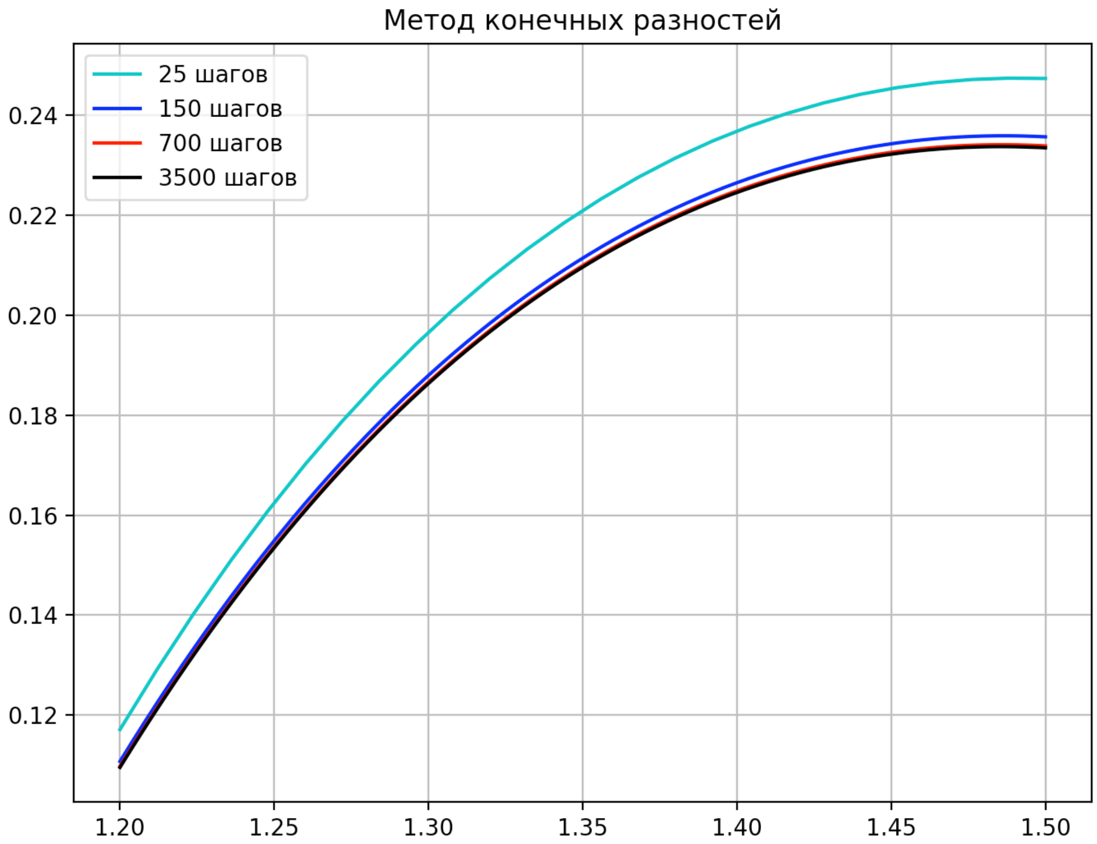
```python
def main():
    # Вариант 8
    # Определяем начальные данные:
    x0 = 1.2
    xn = 1.5
    p = lambda x: 3
    q = lambda x: -1 / x
    f = lambda x: -x - 1
    sigma = [0, 2]
    gamma = [1, -1]
    delta = [1, 0.5]
    de = DifferentialEquation(p, q, f, sigma, gamma, delta)
    boundaryValueProblem(x0, xn, de, [25, 150, 700, 3500])

    # Тест для проверки метода:
    x0 = -1
    xn = 1
    p = lambda x: 2
    q = lambda x: 0
    f = lambda x: 0
    sigma = [1, 1]
    gamma = [0, 1]
    delta = [1, 2]
    de = DifferentialEquation(p, q, f, sigma, gamma, delta)
    yreal = lambda x: (-exp(2 - 2 * x) + 1 + 2 * exp(4)) / (1 + exp(4))
    boundaryValueProblem(x0, xn, de, [5, 15, 45], yreal)
```

Тестирование программы

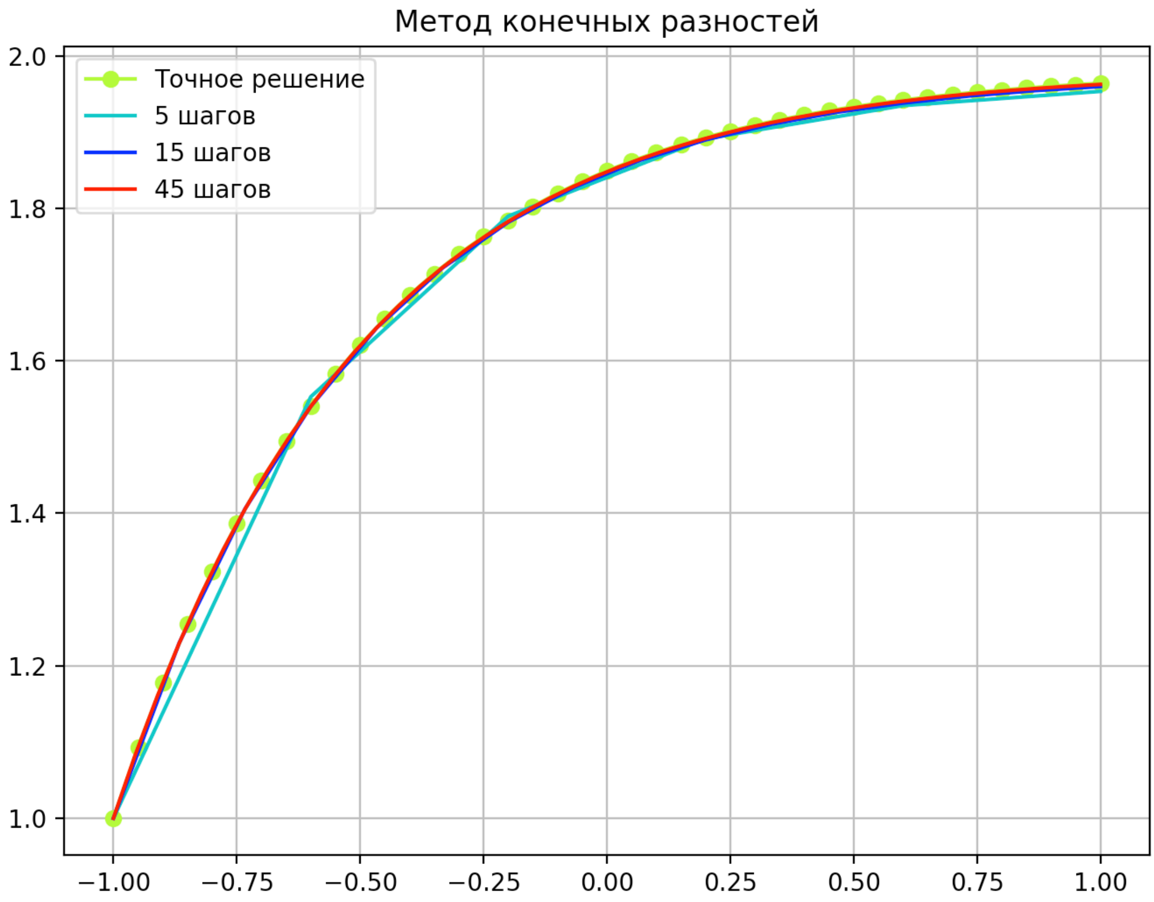
$$y'' + 3y' - \frac{y}{x} = x + 1$$

$y'(1.2) = 1$

$2y(1.5) - y'(1.5) = 0.5.$



$$y''(x) + 2\,y'(x) = 0$$
$$\{y(-1) = 1,\ y'(1) + y(1) = 2\}$$
$$y(x) = \frac{-e^{2-2x} + 1 + 2e^4}{1 + e^4}$$



Выводы

В ходе выполнения поставленной задачи был реализован метод прогонки для решения краевой задачи для дифференциального уравнения второго порядка. Тестирование показало, что с увеличением количества разбиений точность метода возрастает.