Министерство образования Республики Беларусь

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Лабораторная работа 1

"Численные методы решения задачи Коши для ОДУ"

Подготовил:
студент 3 курса 3 группы
Тев Никита Михайлович

Преподаватель:
Горбачёва Юлия
Николаевна

Минск, 2019 г.

1. Постановка задачи

Задачу Коши для данного дифференциального уравнения второго порядка преобразовать к задаче Коши для системы двух дифференциальных уравнений первого порядка. Найти решение последней задачи на отрезке $[a, b]$ с шагом $\tau = 0.05$ указанными методами. Оценить погрешность численного решения с помощью правила Рунге (для одношаговых методов). Сравнить численное решение с известным аналитическим решением $u(t)$.

2. Входные данные

$t^2u''(t)+t^3u'(t)+(t^2-2)u(t)=0,$ $1\le t\le 2 \quad u(1)=1, \quad u'(1)=-1,$ $u(t)=t^{-1}.$	Явный метод Рунге-Кутта 3-го порядка; экстраполяционный метод Адамса 3-го порядка; неявный метод трапеций.

3. Краткие теоретические сведения

Данное ДУ второго порядка преобразуем к задаче Коши для системы двух ДУ первого порядка с помощью следующей замены:

$$\begin{cases} U_1(t) = U(t) \\ U_2(t) = U_1'(t) = U'(t) \end{cases}$$

Получим задачу Коши:

$$\begin{cases} U_1'(t) = f_1(t, U_1(t), U_2(t)) = U_2(t) \\ U_2'(t) = f_2(t, U_1(t), U_2(t)) = -tU_2(t) - \frac{t^2-2}{t^2}U_1(t) \\ U_1(1) = 1,\ U_2(1) = -1 \end{cases}$$

Тогда метод Рунге-Кутта 3 порядка (3 порядок точности) для данной задачи будет иметь следующий вид:

$$\begin{cases} y_{1,j+1} = y_{1,j} + \frac{\tau}{6}(k_1 + 4k_2 + k_3) \\ y_{2,j+1} = y_{2,j} + \frac{\tau}{6}(q_1 + 4q_2 + q_3) \\ k_1 = y_{2,j} \\ q_1 = -t_j y_{2,j} - \frac{t_j^2-2}{t_j^2} y_{1,j} \\ k_2 = y_{2,j} + \frac{1}{2}\tau q_1 \\ q_2 = -(t_j + \frac{1}{2}\tau)(y_{2,j} + \frac{1}{2}\tau q_1) - \frac{(t_j+\frac{1}{2}\tau)^2-2}{(t_j+\frac{1}{2}\tau)^2}(y_{1,j} + \frac{1}{2}\tau k_1) \\ k_3 = y_{2,j} - \tau q_1 + 2\tau q_2 \\ q_3 = -(t_j + \tau)(y_{2,j} - \tau q_1 + 2\tau q_2) - \frac{(t_j+\tau)^2-2}{(t_j+\tau)^2}(y_{1,j} - \tau k_1 + 2\tau k_2) \\ y_{1,0} = 1,\ y_{2,0} = -1,\ j = \overline{0, 19} \end{cases}$$

Экстраполяционный метод Адамса 3 порядка (3 порядок точности):

$$\begin{cases} y_{1,j+1} = y_{1,j} + \frac{\tau}{12}(23y_{2,j} - 16y_{2,j-1} + 5y_{2,j-2}) \\ y_{2,j+1} = y_{2,j} + \frac{\tau}{12}(23(-t_j y_{2,j} - \frac{t_j^2-2}{t_j^2} y_{1,j}) - 16(-t_{j-1} y_{2,j-1} - \frac{t_{j-1}^2-2}{t_{j-1}^2} y_{1,j-1}) + 5(-t_{j-2} y_{2,j-2} - \frac{t_{j-2}^2-2}{t_{j-2}^2} y_{1,j-2}) \\ j = \overline{2, 19} \end{cases}$$

Метод является многостадийным, поэтому значения в первых трёх точках берем из метода Рунге-Кутта.

Неявный метод трапеций (2 порядок точности):

$$\begin{cases} y_{1,j+1} = y_{1,j} + \frac{\tau}{2}(y_{2,j} + y_{2,j+1}) \\ y_{2,j+1} = y_{2,j} + \frac{\tau}{2}(-t_j y_{2,j} - \frac{t_j^2-2}{t_j^2} y_{1,j} - t_{j+1} y_{2,j+1} - \frac{t_{j+1}^2-2}{t_{j+1}^2} y_{1,j+1}) \\ j = \overline{0, 19} \end{cases}$$

В неявном методе трапеций на каждой итерации возникает ситема уравнений относительно $y_{1,j+1}$ и $y_{2,j+1}$, однако, её можно легко разрешить, воспользовавшись методом Гаусса.

4. Листинг программы

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

a, b, step, step2 = (1.0, 2.0, 0.05, 0.1)
u1_0, u2_0 = (1, -1)
n = int((b-a)/step)+1
n2 = int((b-a)/step2)+1

t = np.linspace(a, b, n)
t2 = np.linspace(a, b, n2)
y1_0 = 1/t
y2_0 = -1/(t**2)

y1r = np.zeros(n, dtype = float)
y1a = np.zeros(n, dtype = float)
y1t = np.zeros(n, dtype = float)
y1r_2 = np.zeros(n2, dtype = float)
y1t_2 = np.zeros(n2, dtype = float)
y1r[0], y1a[0], y1t[0], y1r_2[0], y1t_2[0] = u1_0, u1_0, u1_0, u1_0, u1_0

y2r = np.zeros(n, dtype = float)
y2a = np.zeros(n, dtype = float)
y2t = np.zeros(n, dtype = float)
y2r_2 = np.zeros(n2, dtype = float)
y2t_2 = np.zeros(n2, dtype = float)
y2r[0], y2a[0], y2t[0], y2r_2[0], y2t_2[0] = u2_0, u2_0, u2_0, u2_0, u2_0


# ## Part 2: Implementing mathematical methods

def fx(x):
    return (x**2 - 2)/x**2


def f1(t, y1, y2):
    return y2

def f2(t, y1, y2):
    return -t*y2 - ((t**2 - 2)/t**2)*y1


# #### 2.1: Runge–Kutta method

def runge_iteration_y(t, y1, y2, j, step):
    k1 = f1(t[j], y1[j], y2[j])
    q1 = f2(t[j], y1[j], y2[j])
    k2 = f1(t[j] + step/2, y1[j] + k1*step/2, y2[j] + q1*step/2)
    q2 = f2(t[j] + step/2, y1[j] + k1*step/2, y2[j] + q1*step/2)
    k3 = f1(t[j] + step, y1[j] - k1*step + 2*k2*step, y2[j] - q1*step + 2*q2*step)
    q3 = f2(t[j] + step, y1[j] - k1*step + 2*k2*step, y2[j] - q1*step + 2*q2*step)
    return (y1[j] + (step/6)*(k1 + 4*k2 + k3), y2[j] + (step/6)*(q1 + 4*q2 + q3))
```

```
# #### 2.2: Adams' method

def adams_iteration_y1(t, y1, y2, j, step):
    return y1[j] + (step/12)*(23 * f1(t[j], y1[j], y2[j]) -
                              16 * f1(t[j-1], y1[j-1], y2[j-1]) +
                              5 * f1(t[j-2], y1[j-2], y2[j-2]))



def adams_iteration_y2(t, y1, y2, j, step):
    return y2[j] + (step/12)*(23 * f2(t[j], y1[j], y2[j]) -
                              16 * f2(t[j-1], y1[j-1], y2[j-1]) +
                              5 * f2(t[j-2], y1[j-2], y2[j-2]))



# #### 2.3: Trapezium method

def trapezium_iteration(t, y1, y2, j, step):
    a1 = step/2
    b1 = y1[j] + step*y2[j]/2
    a2 = step*t[j+1]/2
    b2 = y2[j] - step*t[j]*y2[j]/2 - step*fx(t[j])*y1[j]/2
    c2 = step*fx(t[j+1])/2

    x2 = (b2 - c2*b1)/(1 + a2 + c2*a1)
    x1 = b1 + a1*x2
    return (x1, x2)


# ## Part 3: Value table calculation

for j in range(1, n):
    y1r[j],y2r[j] = runge_iteration_y1(t, y1r, y2r, j-1, step)

for j in range(1, n2):
    y1r_2[j], y2r_2[j] = runge_iteration_y1(t2, y1r_2, y2r_2, j-1, step2)

for j in range(1, 3):
    y1a[j] = y1r[j]
    y2a[j] = y2r[j]

for j in range(3, n):
    y1a[j] = adams_iteration_y1(t, y1a, y2a, j-1, step)
    y2a[j] = adams_iteration_y2(t, y1a, y2a, j-1, step)

for j in range(1, n):
    y1t[j], y2t[j] = trapezium_iteration(t, y1t, y2t, j-1, step)

for j in range(1, n2):
    y1t_2[j], y2t_2[j] = trapezium_iteration(t2, y1t_2, y2t_2, j-1, step2)


# ## Part 4: Error calculation
```

```
err_r_1 = max(max(np.absolute(y1r[::2] - y1r_2)), max(np.absolute(y2r[::2] - y2r_2))) / 7

err_t_1 = max(max(np.absolute(y1t[::2] - y1t_2)), max(np.absolute(y2t[::2] - y2t_2))) / 3

err_r_2 = max(max(np.absolute(y1_0 - y1r)), max(np.absolute(y2_0 - y2r)))

err_a_2 = max(max(np.absolute(y1_0 - y1a)), max(np.absolute(y2_0 - y2a)))

err_t_2 = max(max(np.absolute(y1_0 - y1t)), max(np.absolute(y2_0 - y2t)))

# ## Part 5: Export results

d = {'t_j':t, 'y1_0':y1_0, 'y2_0':y2_0, 'y1r':y1r, 'y2r':y2r, 'y1a':y1a, 'y2a':y2a, 'y1t':y1t, 'y2t':y2t}

df = pd.DataFrame(data = d)
df = df.to_excel("output2.xlsx")
```

5. Результаты работы

j	t_j	Точное решение		Рунге-Кутта (3п)		Адамса (3п)		Трапеций (2п)	
		u(tj)	*u'(tj)*	*y1(tj)*	*y2(tj)*	*y1(tj)*	*y2(tj)*	*y1(tj)*	*y2(tj)*
0	1,00	1,000000	-1,000000	1,000000	-1,000000	1,000000	-1,000000	1,000000	-1,000000
1	1,05	0,952381	-0,907029	0,952379	-0,907026	0,952379	-0,907026	0,952330	-0,906815
2	1,10	0,909091	-0,826446	0,909087	-0,826441	0,909087	-0,826441	0,909007	-0,826075
3	1,15	0,869565	-0,756144	0,869561	-0,756137	0,869522	-0,755951	0,869464	-0,755661
4	1,20	0,833333	-0,694444	0,833328	-0,694436	0,833276	-0,694132	0,833225	-0,693883
5	1,25	0,800000	-0,640000	0,799994	-0,639991	0,799935	-0,639600	0,799894	-0,639385
6	1,30	0,769231	-0,591716	0,769224	-0,591706	0,769166	-0,591259	0,769132	-0,591069
7	1,35	0,740741	-0,548697	0,740734	-0,548686	0,740683	-0,548205	0,740655	-0,548032
8	1,40	0,714286	-0,510204	0,714279	-0,510193	0,714239	-0,509693	0,714216	-0,509535
9	1,45	0,689655	-0,475624	0,689648	-0,475613	0,689622	-0,475107	0,689603	-0,474960
10	1,50	0,666667	-0,444444	0,666660	-0,444433	0,666650	-0,443931	0,666634	-0,443792
11	1,55	0,645161	-0,416233	0,645155	-0,416222	0,645161	-0,415731	0,645150	-0,415600
12	1,60	0,625000	-0,390625	0,624994	-0,390614	0,625018	-0,390139	0,625009	-0,390015
13	1,65	0,606061	-0,367309	0,606054	-0,367299	0,606096	-0,366844	0,606091	-0,366726
14	1,70	0,588235	-0,346021	0,588229	-0,346010	0,588289	-0,345578	0,588286	-0,345466
15	1,75	0,571429	-0,326531	0,571423	-0,326520	0,571499	-0,326113	0,571499	-0,326007
16	1,80	0,555556	-0,308642	0,555550	-0,308632	0,555643	-0,308251	0,555645	-0,308151
17	1,85	0,540541	-0,292184	0,540535	-0,292174	0,540643	-0,291820	0,540648	-0,291726
18	1,90	0,526316	-0,277008	0,526311	-0,276999	0,526433	-0,276672	0,526441	-0,276583
19	1,95	0,512821	-0,262985	0,512816	-0,262976	0,512952	-0,262676	0,512961	-0,262593
20	2,00	0,500000	-0,250000	0,499995	-0,249991	0,500144	-0,249718	0,500155	-0,249640

	Метод		
	Рунге-Кутта	Адамса	Трапеций
Оценка погрешности по правилу Рунге	0,0000124	-	0,000671
Оценка по модулю	0,0000111	0,00052	0,000668

6. Выводы

Как можно увидеть из оценки погрешностей, рассмотренные методы действительно обеспечивают заявленные порядки точности при нахождении приближенного решения задачи Коши для ОДУ.