Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа №7

Решение ОДУ

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Общие функции для ЛР на Python

```python
import numpy as np

# constants
a = np.array([
    [0, 0, 0, 0],
    [1/2, 0, 0, 0],
    [0, 1/2, 0, 0],
    [0, 0, 1, 0]
])
b = np.array([1/6, 1/3, 1/3, 1/6])
c = np.array([0, 1/2, 1/2, 1])


def extrapolate_adams(f, y0, t0, t_end, h):
    n = int((t_end - t0[0]) / h)
    y = np.empty(n + 1)
    t = np.empty(n + 1)
    y[:2] = y0
    t[:2] = t0
    for i in range(1, n):
        y[i + 1] = y[i] + (h / 2) * (3 * f(t[i], y[i]) - f(t[i - 1], y[i - 1]))
    return y
```

```
def eyler(f, y0, t0, t_end, h):
    n = int((t_end - t0) / h)
    y = np.empty(n + 1)
    y[0] = y0
    for i in range(n):
        y[i + 1] = (y[i] + h * f(t0 + i * h, y[i]))
    return y


def eyler_modified_step(f, y0, t0, h):
    y_pred = y0 + h * f(t0, y0)
    y_corr = y0 + h * (f(t0, y0) + f(t0 + h, y_pred)) / 2
    return y_corr


def rk4_step(f, t, s, h):
    ss = 4
    k = [f(t, s)]
    for i in range(1, ss):
        diff = s
        for j in range(i):
            diff += h * a[i, j] * k[j]
        k.append(f(t + c[i] * h, diff))
    diff = s
    for i in range(ss):
        diff += h * b[i] * k[i]
    return diff


def rk4_nsteps(f, y0, t0, t_end, h):
    n = int((t_end - t0) / h)
    arr = np.empty((n + 1, 2))
    arr[:, 0] = np.linspace(t0, t_end, n + 1, endpoint=True)
    arr[0, 1] = y0

    for i in range(n):
        arr[i + 1, 1] = rk4_step(f,           # right part of SODE
                                 arr[i, 0],   # t_0
                                 arr[i, 1],   # s_0
                                 h)           # time step
    return arr[:, 1]


def runge_error_step(solver_step, f, y0, t0, h, p):
    y_h = solver_step(f, y0, t0, h)
    y_1_h2 = solver_step(f, y0, t0, h / 2)
    y_2_h2 = solver_step(f, y_1_h2, t0 + h / 2, h / 2)
    error = (y_h - y_2_h2) / (2 ** p - 1)
    return error
```

```python
def eyler_modified(f, y0, t0, t_end, h):
    n = int((t_end - t0) / h)
    y = np.empty(n + 1)
    y[0] = y0
    for i in range(n):
        y[i + 1] = eyler_modified_step(f, y[i], t0 + i * h, h)
    return y


def eyler_adaptive(f, y0, t0, t_end, h0, tol):
    ys = [y0]
    ts = [t0]
    y_prev = y0
    t, h = t0, h0
    while t < t_end:
        y = eyler_modified_step(f, y_prev, t, h)
        err = abs(runge_error_step(eyler_modified_step, f, y_prev, t, h, 2))
        factor = min(max(tol / np.sqrt(2 * err), 0.3), 2)
        if factor >= 1:
            ys.append(y)
            y_prev = y
            t += h
            ts.append(t)
        h = 0.9 * h * factor
    return ys, ts


def runge_error(solver, f, y0, t0, t_end, h, p):
    normal_precision = solver(f, y0, t0, t_end, h)
    double_precision = solver(f, y0, t0, t_end, h / 2)
    errors = (double_precision[::2] - normal_precision) / (2 ** p - 1)
    return errors
```

2 Решение ОДУ и оценка погрешности методом Рунге (7.1.16)

2.1 Формулировка задачи

Найти приближенное решение задачи Коши для ОДУ 1 порядка. Оценить погрешность решения.

1. Задать исходные данные: функцию f правой части, начальное значение y_0.
2. Найти приближенное решение задачи Коши с шагом $h = 0.1$ по явному методу Эйлера.
3. Написать программу для поиска приближенного решения с шагом $h = 0.1$ по методу Рунге-Кутты 4 порядка точности.

4. Найти решение аналитически.

5. Построить таблицы значений приближенных и точного решений, построить графики.

6. Оценить погрешность двумя способами: по формуле $\varepsilon = max|y(t_i) - y_i|$ и по правилу Рунге (по правилу двойного пересчета).

7. Выяснить, при каком h^* решение по методу Эйлера имеет такую же погрешность, как решение при помощи метода Рунге-Кутты с шагом $h = 0.1$.

2.2 Вариант

$$f(t, y) = -\frac{y}{t} + 3t$$

$$t_0 = 1; T = 2; y_0 = 1$$

2.3 Аналитическое решение

Дано ОДУ:

$$\frac{dy}{dt} = -\frac{y}{t} + 3t$$

Сначала решим уравнение:

$$\frac{dy}{dt} + \frac{y}{t} = 0$$

Разделим переменные и получим общее решение:

$$\frac{t}{dt} = -\frac{y}{dy}$$

$$y(t) = \frac{C}{t}$$

Полное решение:

$$\frac{C'}{t} + (-\frac{C}{t^2}) + \frac{C}{t^2} = 3t$$

$$C' = 3t^2$$

$$C = t^3 + C_2$$

$$y = t^2 + t^{-1}$$

$$y(1) = 1$$

$$y = t^2$$

2.4 Код на Python

```python
import numpy as np
import matplotlib.pyplot as plt
from numba import njit

from ode import eyler, rk4_nsteps, runge_error

@njit
def f(t, y):
    return - y / t + 3 * t


@njit
def analytical_solution(t):
    return t ** 2


t0, t_end, y0 = 1, 2, 1
h = 0.1
N = int((t_end - t0) / h)

t_values = np.array([t0 + i * h for i in range(N + 1)])

analytical_values = np.array([analytical_solution(t) for t in t_values])
eyler_values = np.array(eyler(f, y0, t0, t_end, h))

rk_values = rk4_nsteps(f, y0, t0, t_end, h)


plt.plot(t_values, eyler_values, label='Eyler')
plt.plot(t_values, rk_values, label='Runge-Kutta')
plt.plot(t_values, analytical_values, label='True solution', ls='--')
plt.legend()
plt.savefig('plots/eyler_vs_rk.png', dpi=300)

eyler_error = np.abs(eyler_values - analytical_values).max()
rk_error = np.abs(rk_values - analytical_values).max()
eyler_runge_error = np.abs(runge_error(eyler, f, y0, t0, t_end, h, 1)).max()
rk_runge_error = np.abs(runge_error(rk4_nsteps, f, y0, t0, t_end, h, 4)).max()

print(f'Eyler Absolute error = {eyler_error}')
print(f'RK4 Absolute error = {rk_error}')
print()
print(f'Eyler Runge error = {eyler_runge_error}')
print(f'RK4 Runge error = {rk_runge_error}')
print()


for i in range(8):
    h /= 10
```

```
    N = int((t_end - t0) / h)
    t_values = np.linspace(t0, t_end, N + 1)

    eyler_values = np.array(eyler(f, y0, t0, t_end, h))
    analytical_values = analytical_solution(t_values)

    eyler_error = np.abs(eyler_values - analytical_values).max()
    if rk_error >= eyler_error:
        break


print(f'For h = {h} Eyler:')
print(f'N = {N} points')
print(f'Eyler method error = {eyler_error}')
```

2.5 Вывод программы

За приемлемое число итераций и время работы достигнуть того же уровня точности, как в методе Рунге-Кутты, не получилось. Максимально достигнутая точность представлена в выводе ниже:

```
Eyler Absolute error = 0.07631578947368389
RK4 Absolute error = 1.3322676295501878e-15

Eyler Runge error = 0.03849527665317165
RK4 Runge error = 1.1842378929335003e-16

For h = 1.0000000000000003e-09 Eyler:
N = 999999999 points
Eyler method error = 4.7500057220872804e-09
```

2.6 Графики результирующих решений

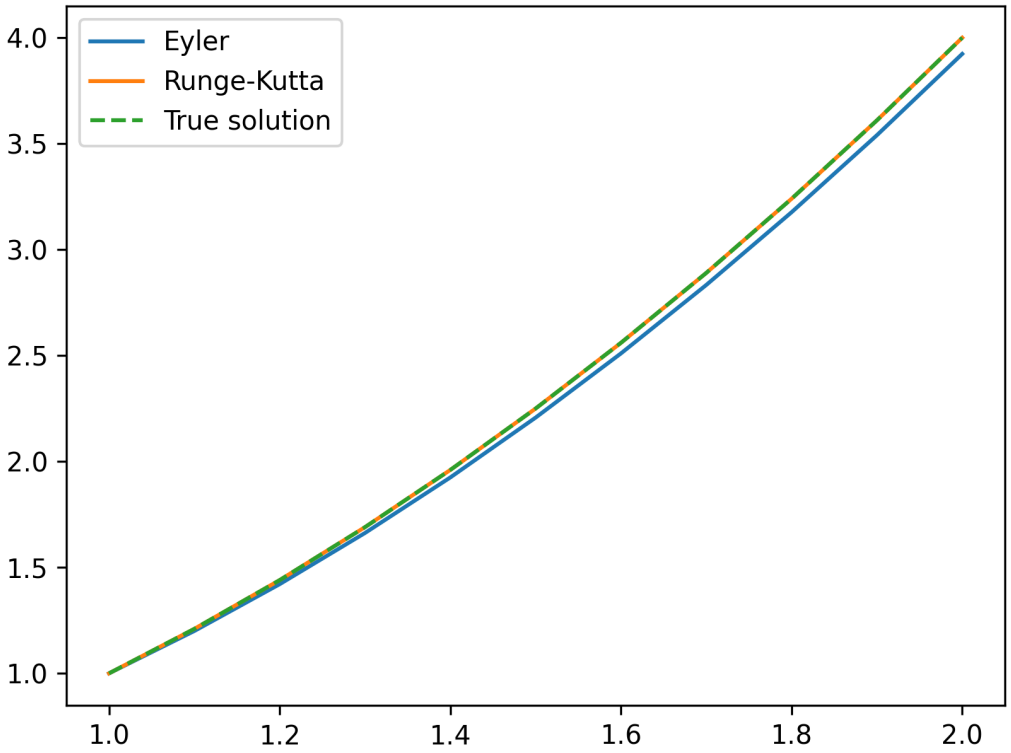

Рис. 1: Графики решений уравнения

3 Уточненное решение ОДУ по методу Рунге (7.3.6)

3.1 Формулировка задачи

Решить приближенно задачу Коши для ОДУ 1 порядка, используя метод Рунге-Кутты 4 порядка и метод в варианте с использованием шагов $h, \frac{h}{2}$. Оценить погрешность по правилу Рунге, вычислить уточненное решение. Построить графики приближенных решений и графики уточненных решений.

3.2 Вариант

Метод - Экстраполяционный метод Адамса 3 порядка.

$$f(t, y) = -ty + (t - 1)e^t y^2$$

$$t_0 = 0; T = 1; y_0 = 1$$

3.3 Код на Python

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from numba import njit

from ode import extrapolate_adams, rk4_nsteps, runge_error


@njit
def f(t, y):
    return - (t * y) + (t - 1) * np.exp(t) * (y ** 2)


t0, t_end, y0 = 0, 1, 1
h = 0.1
n = int(2 * (t_end - t0) / h)

t_values = np.array([t0 + i * h / 2 for i in range(n + 1)])[::2]
rk_solution = rk4_nsteps(f, y0, t0, t_end, h / 2)[::2]
rk_errors = runge_error(rk4_nsteps, f, y0, t0, t_end, h, 4)

y0_adams = rk_solution[:2]
t0_adams = t_values[:2]

adams_solution = extrapolate_adams(f, y0_adams, t0_adams, t_end, h / 2)[::2]
adams_errors = runge_error(extrapolate_adams, f, y0_adams, t0_adams, t_end, h, 2)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))
ax[0].plot(t_values, rk_solution, label='Base RK4', color='green')
ax[0].plot(t_values, rk_solution + rk_errors, label='Precise RK4', color='blue')
ax[1].plot(t_values, adams_solution, label='Base Adams')
ax[1].plot(t_values, adams_solution + adams_errors, label='Precise Adams')
fig.legend()
plt.savefig('plots/adams_vs_rk.png', dpi=300)
```

3.4 Графики результирующих решений

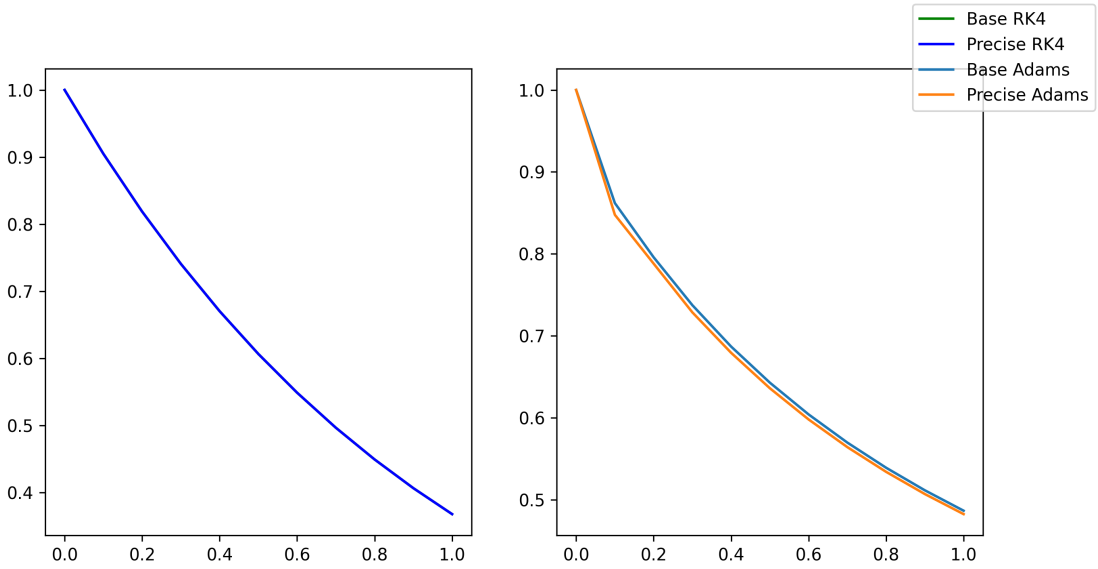

Рис. 2: Графики решений уравнения

4 Уточненное решение ОДУ по методу Рунге (7.3.6)

4.1 Формулировка задачи

Решить приближенно задачу Коши для ОДУ 1 порядка с помощью метода в индивидуальном варианте с точностью $\varepsilon = 10^{-4}$, использовать алгоритм автоматического выбора шага. В результате работы должен создаваться файл с вектором значений приближенного решения, а также значения шага h, при котором достигается точность. Программа по запросу должна выдавать таблицу значений или график найденного решения.

4.2 Вариант

Метод - Модифицированный метод Эйлера 2 порядка.

$$f(t, y) = -\frac{1}{3}y\sqrt{t} + \frac{2}{3}y^2 sint$$

$$t_0 = 2; T = 10; y_0 = 2.2$$

4.3 Код на Python

```
import numpy as np
from numba import njit
import matplotlib.pyplot as plt

from ode import eyler_adaptive, eyler_modified


@njit
```

```python
def f(t, y):
    return - 1/3 * y * np.sqrt(t) + 2/3 * (y ** 2) * np.sin(t)


t0, t_end, y0 = 2, 10, 2.2
h0 = 0.05
n = int((t_end - t0) / h0)

ys_eyler_m, ts_eyler_m = eyler_adaptive(f, y0, t0, t_end, h0, 10e-4)
ys_eyler_m = np.array(ys_eyler_m)
ts_eyler_m = np.array(ts_eyler_m)

ys_eyler = eyler_modified(f, y0, t0, t_end, h0)
ts_eyler = np.linspace(t0, t_end, n + 1, endpoint=True)

np.save('eyler_adaptive_values.npy', ys_eyler)
np.save('eyler_adaptive_points.npy', ts_eyler)
hs = ts_eyler[1:] - ts_eyler[:-1]
np.save('eyler_adaptive_steps.npy', hs)

print('Enter option: 1 - table, 0 - plot')
opt = int(input())
if opt:
    for t, y in zip(ts_eyler_m, ys_eyler_m):
        print(f't = {t:.4f}, y = {y:.4f}')
else:
    plt.plot(ts_eyler_m, ys_eyler_m, label='Adaptive Eyler')
    plt.plot(ts_eyler, ys_eyler, label='Eyler')
    plt.legend()
    plt.savefig('plots/adaptive_step.png', dpi=300)
    plt.show()
```

4.4 Вывод программы

```
Enter option: 1 - table, 0 - plot
t = 2.0000, y = 2.2000
t = 2.0098, y = 2.2187
t = 2.0213, y = 2.2410
t = 2.0318, y = 2.2617
...
t = 9.4341, y = 0.0076
t = 9.5380, y = 0.0068
t = 9.6450, y = 0.0061
t = 9.7552, y = 0.0055
t = 9.8690, y = 0.0049
t = 9.9865, y = 0.0043
t = 10.1082, y = 0.0038
```

4.5 Графики результирующих решений

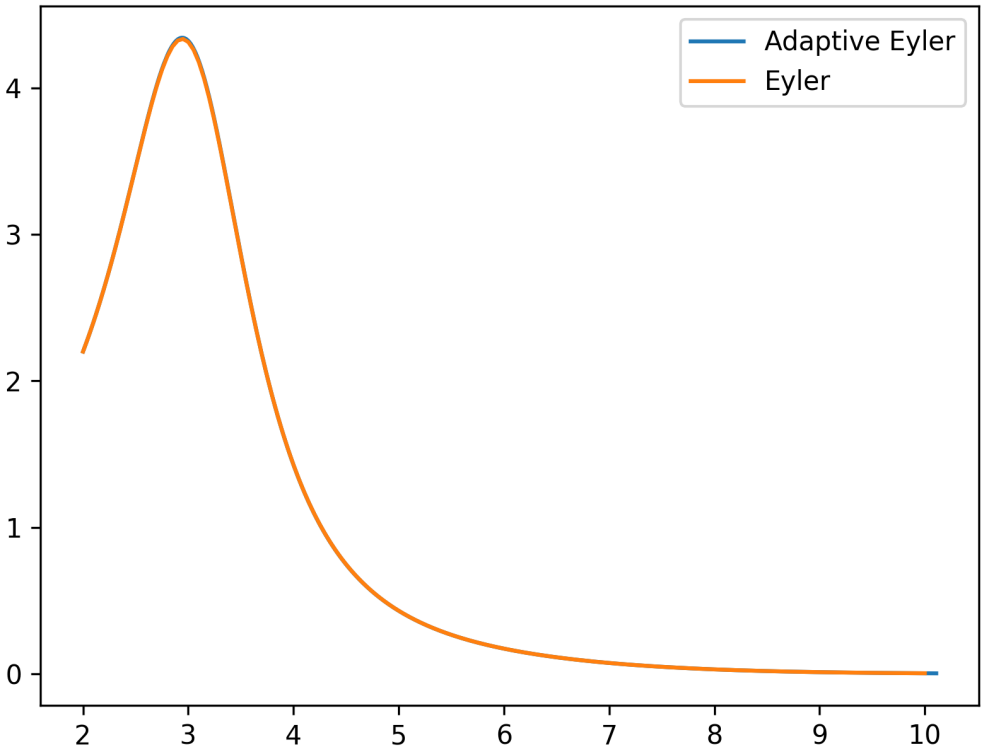

Рис. 3: Графики решений уравнения