Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчет по лабораторной работе №3**

**Дисциплина: «**Практикум по вычислительной математике».

Выполнил

студент гр. 5130901/20003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Вагнер

(подпись)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Цыган

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург   
2024

**ЗАДАНИЕ**

**Вариант 24-A (п. 1 и 10):**

Привести дифференциальное уравнение к системе двух уравнений первого порядка.

Решить на интервале

Начальные условия:

Точное решение:

*1)* используя программу RKF45 с шагом печати и выбранной погрешностью EPS в диапазоне 0.0001–0.000001;

*10)* используя явный метод ломаных Эйлера;

Сравнить результаты, полученные заданными приближенными способами, с точным решением.

Исследовать влияние величины шага интегрирования **hint** на величины **локальной** и **глобальной** погрешностей решения заданного уравнения для чего решить уравнение, используя **3** значения шага интегрирования, существенно меньшие исходной величины **0.1** (например, **hint = 0.05**; **hint = 0.025**; **hint = 0.0125**).

**ХОД РАБОТЫ**

Для выполнения лабораторной работы был использован язык программирования Python, а также библиотеки SciPy и NumPy.

Первым пунктом работы было приведение дифференциального уравнения к системе из двух дифференциальных уравнений первого порядка.

Таким образом, путём замены была получена следующая система:

Для реализации RKF45 был использован модуль integrate библиотеки SciPy.

Для реализации явного метода ломаных Эйлера использовалась следующая формула: .

RKF45

integratorName = 'dopri5'  
eps = 1e-5  
h\_print = 0.1  
intervalStart = 1  
intervalEnd = 2  
  
def RKF45(f, T, X0):  
 r = integrate.ode(f).set\_integrator(integratorName, atol=eps).set\_initial\_value(X0, T[0])  
  
 X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
  
 for i in range(1, len(T)):  
 X[i] = r.integrate(T[i])  
  
 if not r.successful():  
 raise RuntimeError('Integration unsuccessful')  
  
 return X

Используя библиотеку SciPy для решения интегральных уравнений требуется использовать следующие метода:

*.integrate.ode(f) –* задать функцию f*;*

*.set\_integrator(integratorName, atol=eps) –* задать программу, которая будет решать уравнение (в нашем случае dopri5, он использует метод RKF45) и погрешность (в нашем случае 1e-5);

*.set\_initial\_value(X0, T[0])* – задать множество начальных значений.

T и X0 – массивы, хранящие множество точек, в которых ищем решение, и множество начальных значений соответственно.

Далее мы создаём матрицу X *,* в которой будут хранится решения ОДУ.

В случае, если решение не удалось, программа остановит исполнение с текстом “'Integration unsuccessful”

EULER

def euler(f, T, X0):  
 X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
 for i in range(1, len(T)):  
 f1 = lambda Y: f(T[i], Y)  
  
 def equations(Y):  
 return [  
 X[i - 1][j] + (T[i] - T[i - 1]) \* f1(Y-1)[j] - Y[j] for j in range(len(Y))  
 ]  
  
 root = fsolve(equations, [0] \* len(X0), xtol=1e-14, maxfev=2 \*\* 30)  
 for j in range(len(X[i])):  
 X[i][j] = X[i - 1][j] + (T[i] - T[i - 1]) \* f1(root)[j]  
  
 return X

Реализация явного метода ломанных Эйлера во многих аспектах повторяет таковую для RKF45. Функция *fsolve()* реализует поиск корней функции. В данном случае функция задана внутри метода *equations(Y)*. Таки образом строка *X[i][j] = X[i - 1][j] + (T[i] - T[i - 1]) \* f1(root)[j]* представляет из себя ранее описанную формулу .

Результаты работы программы

Для упрощения проведения анализа результатов посчитаем внутри программы среднее арифметическое погрешностей для каждого из h.

Рассмотрим результат работы программы.

Изображение выглядит как текст, снимок экрана, меню, Шрифт

Автоматически созданное описание

Таблица результатов работы программы для h = 0.1

|  |  |  |  |
| --- | --- | --- | --- |
| t | RKF45 | Euler | Точн. значение |
| 1.0 | 1.0000000000000000 | 1.0000000000000000 | 1.000000 |
| 1.1 | 1.3310000062541572 | 1.3104347826086959 | 1.331000 |
| 1.2 | 1.7280000281806742 | 1.6952587917324133 | 1.728000 |
| 1.3 | 2.1970000393179747 | 2.1593598773702625 | 2.197000 |
| 1.4 | 2.7440000681800867 | 2.7081943612170880 | 2.744000 |
| 1.5 | 3.3750000992141231 | 3.3475764622895952 | 3.375000 |
| 1.6 | 4.0960001329554379 | 4.0835522355494591 | 4.096000 |
| 1.7 | 4.9130001698580719 | 4.9223216680066946 | 4.913000 |
| 1.8 | 5.8320002103206532 | 5.8701891828582680 | 5.832000 |
| 1.9 | 6.8590002547031110 | 6.9335314170660229 | 6.859000 |
| 2.0 | 8.0000003033377656 | 8.1187757866890724 | 8.000000 |

Составим графики изменения погрешностей для обоих методов в зависимости от t.

Таблица зависимости средней погрешности от h.

|  |  |  |
| --- | --- | --- |
| h | Средн. погр. RKF45 | Средн. погр. Euler |
| 0.1 | 0.0000001193020011 | 0.0370401403502295 |
| 0.05 | 0.0000000077718274 | 0.0164026587613379 |
| 0.025 | 0.0000000003520690 | 0.0080934151477602 |
| 0.0125 | 0.0000000000116123 | 0.0040747745017595 |

Представим те же результаты в виде графиков.

Рассмотрим локальные погрешности каждого из методов в зависимости от h:

|  |  |  |
| --- | --- | --- |
| h | Лок. погр. RKF45 | Лок. погр. Euler |
| 0.1 | 0.0000000062541567 | 0.0205652173913045 |
| 0.05 | 0.0000000000299676 | 0.0062456896551724 |
| 0.025 | 0.0000000000134928 | 0.0017115205223881 |
| 0.0125 | 0.0000000000002238 | 0.0004478198298055 |

Представим те же результаты в виде графиков.

Рассмотрим результаты решения уравнения для всех длин шагов для первого и последнего шага.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| h | t | RKF45 | Euler | Точн. |
| 0.1 | 1.1000 | 1.3310000062541572 | 1.3104347826086959 | 1.331000 |
| 2.0000 | 8.0000003033377656 | 8.1187757866890724 | 8.000000 |
| 0.05 | 1.0500 | 1.1576250000299677 | 1.1513793103448278 | 1.157625 |
| 2.0000 | 8.0000000189076061 | 8.0428416160645817 | 8.000000 |
| 0.025 | 1.0250 | 1.0768906250134926 | 1.0751791044776118 | 1.076891 |
| 2.0000 | 8.0000000008352643 | 8.0177606720595751 | 8.000000 |
| 0.0125 | 1.0125 | 1.0379707031252237 | 1.0375228832951944 | 1.037971 |
| 2.000 | 8.0000000000275602 | 8.0080176484883445 | 8.000000 |

Вывод

Из результатов очевидно, что погрешность вычислений метода RKF45 намного меньше таковых для явного метода ломанных Эйлера, хотя ни один из них не смог уложиться в заданную погрешность *1е-5.* Локальные погрешности обоих методов стремительно возрастают к концу отрезка, но примечательно то, что минимум погрешности RKF45 находится в самом начале отрезка, а Euler примерно в 1/3 промежутка. Также нетрудно заметить, что глобальная погрешность кратно уменьшается при уменьшении шага интегрирования, хотя RKF45 тут тоже показал лучший результат с более стремительным уменьшением глобальной погрешности. Кроме того, очевидно, что локальная погрешность уменьшается с приближением к заданному краевому значению, то есть уменьшению первого шага, а графики зависимости локальной погрешности от шага напоминают ветвь параболы.

Приложение

import rkf45  
import forwardEuler  
import numpy as np  
import scipy.integrate as integrate  
from scipy.optimize import fsolve  
  
def preciseY(t):  
 return t \* t \* t  
  
integratorName = 'dopri5'  
eps = 1e-5  
h\_print = 0.1  
intervalStart = 1  
intervalEnd = 2  
  
def RKF45(f, T, X0):  
 r = integrate.ode(f).set\_integrator(integratorName, atol=eps).set\_initial\_value(X0, T[0])  
  
 X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
  
 for i in range(1, len(T)):  
 X[i] = r.integrate(T[i])  
  
 if not r.successful():  
 raise RuntimeError('Integration unsuccessful')  
  
 return X  
  
  
def euler(f, T, X0):  
 X = np.zeros((len(T), len(X0)))  
 X[0] = X0  
 for i in range(1, len(T)):  
 f1 = lambda Y: f(T[i], Y)  
  
 def equations(Y):  
 return [  
 X[i - 1][j] + (T[i] - T[i - 1]) \* f1(Y-1)[j] - Y[j] for j in range(len(Y))  
 ]  
  
 root = fsolve(equations, [0] \* len(X0), xtol=1e-14, maxfev=2 \*\* 30)  
 for j in range(len(X[i])):  
 X[i][j] = X[i - 1][j] + (T[i] - T[i - 1]) \* f1(root)[j]  
  
 return X  
  
  
def f(t, X):  
 dX = np.zeros(len(X))  
 dX[0] = X[1]  
 dX[1] = (6 \* X[0]) / (t \*\* 2)  
 return dX  
  
  
for h in 0.1, 0.05, 0.025, 0.0125:  
 print(f"{h = }:")  
 rng = np.arange(intervalStart, intervalEnd + h - 1e-9, h)  
 steps = len(rng)  
 X0 = [1, 3]  
  
 res\_rk45 = [i[0] for i in RKF45(f, rng, X0)]  
 res\_be = [i[0] for i in euler(f, rng, X0)]  
 res\_precise = [preciseY(rng[i]) for i in range(steps)]  
  
 if h == 0.1:  
 print("t\t\t rk45\t\t\t\t euler\t\t\t\t precise")  
 for i in range(steps):  
 print(f"{rng[i]:.4f}\t {res\_rk45[i]:.16f}\t {res\_be[i]:.16f}\t {res\_precise[i]:.6f}")  
 else:  
 print(f"{rng[1]:.4f}\t {res\_rk45[1]:.16f}\t {res\_be[1]:.16f}\t {res\_precise[1]:.6f}")  
 print(f"{rng[-1]:.4f}\t {res\_rk45[-1]:.16f}\t {res\_be[-1]:.16f}\t {res\_precise[-1]:.6f}")  
  
 rk45Error = sum([abs(res\_rk45[i] - res\_precise[i]) for i in range(steps)]) / steps  
 eulerError = sum([abs(res\_be[i] - res\_precise[i]) for i in range(steps)]) / steps  
  
 print(f"Local {abs(res\_rk45[1] - res\_precise[1]):.16f}\t {abs(res\_be[1] - res\_precise[1]):.16f}")  
 print(f"Average {rk45Error:.16f}\t {eulerError:.16f}")  
 print("\n")