Белорусский государственный университет
Факультет прикладной математики и информатики

Методы решения граничной задачи для ОДУ-1

Отчет по лабораторной работе
студента 3 курса 3 группы
Аквуха Джеймса

Преподаватель:
Будник А.М.

1. Постановка задачи

Поставлена задача Коши для одного уравнения и системы 2 уравнений:

$$\begin{cases} y' = f(x, y) \\ y(a) = y_0, \quad x \in [a, b] \end{cases}$$

$$\begin{cases} y' = f_1(x, y, z) \\ z' = f_2(x, y, z) \\ y(a) = y_0, \quad z(a) = z_0, \quad x \in [a, b] \end{cases}$$

Требуется найти значения искомых функций на отрезке [a, b] на равномерной сетке из N + 1 точки.

2. Методы решения

2.1 Метод рядов

Искомая функция приближается своим разложением в ряд Тейлора в окрестности узловых точек. Точки вычисляются последовательно, начиная с точки, для которой известно точное значение из постановки задачи Коши:

$$y_k(x_{n+1}) \approx \sum_{i=0}^{k} \frac{(x_{n+1} - x_n)^i}{i!} y^{(i)}(x_n), \quad n = \overline{0, N-1}$$

Для системы уравнений записываются два аналогичных приближения искомых функций.

2.2 Явный метод Эйлера

Изменение значения искомой функции приближается с помощью формулы левых прямоугольников.

$$y_{n+1} = y_n + hf(x_n, y_n), \quad y(x_0) = y_0, \quad h = \frac{b-a}{N}$$

Для системы уравнений записываются два аналогичных приближения искомых функций.

2.3 Неявный метод Эйлера

Изменение значения искомой функции приближается с помощью формулы правых прямоугольников. Полученная система решается методом Ньютона.

$$y_{n+1}^{(k+1)} = \frac{y_n + hf(x_{n+1}, y_{n+1}^{(k)}) - hy_{n+1}^{(k)} f_y(x_{n+1}, y_{n+1}^{(k)})}{1 - hf_y(x_{n+1}, y_{n+1}^{(k)})}$$

Начальное приближение для метода Ньютона находится по формуле явного метода Эйлера. Итерации проводятся пока норма невязки изменения решения не окажется меньше ε.

$$y_{n+1}^{(0)} = y_n + hf(x_n, y_n)$$

Для системы уравнений записываются два аналогичных приближения искомых функций.

$$y_{n+1}^{(k+1)} = \frac{y_n + hf_1(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)}) - hy_{n+1}^{(k)}(f_1)_y(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)})}{1 - h(f_1)_y(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)})}$$

$$z_{n+1}^{(k+1)} = \frac{z_n + hf_2(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)}) - hz_{n+1}^{(k)}(f_2)_z(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)})}{1 - h(f_2)_z(x_{n+1}, y_{n+1}^{(k)}, z_{n+1}^{(k)})}$$

$$y(x_0) = y_0, \quad z(x_0) = z_0, \quad h = \frac{b-a}{N}$$

2.4 Метод предиктор-корректор

Изменение искомой функции приближается квадратурной формулой:

$$y(x_{n+1}) \approx y(x_n) + h\sum_{i=0}^{q} A_i Z_n(\alpha_i),$$

Требуя, чтобы формула была точна для всех многочленов степени $2q+1$, получаем систему уравнений:

$$\sum_{i=0}^{q} A_i \alpha_i^j = \frac{1}{j+1}, j = \overline{0, 2q+1}$$

, решая которую для $q = 2$, получаем:

$$y_{n+1/4}^{[2]} = y_n^{[4]} + \frac{h}{4} f_n^{[4]}$$

$$y_{n+1/2}^{[3]} = y_n^{[4]} + \frac{h}{2} f_{n+1/4}^{[4]}$$

$$y_{n+1}^{[3]} = y_n^{[4]} + hf_{n+1/2}^{[3]}$$

$$y_{n+1}^{[4]} = y_n^{[4]} + \frac{h}{6}\left(f_n^{[4]} + 4f_{n+1/2}^{[3]} + f_{n+1}^{[3]}\right)$$

В случае системы полученная система рассматривается для обоих уравнений.

2.5 Метод Рунге-Кутты

Правило Рунге-Кутты основано на приближении изменения искомой функции некоторой линейной комбинацией.

$$y(x_{n+1}) = y(x_n) + h\int_0^1 f(x_n + \alpha h, y(x_n + \alpha h))d\alpha$$

$$\phi_0 = hf(x, y(x))$$

$$\phi_i = hf\left(x + \alpha_i h, y(x) + \sum_{j=0}^{i-1} \beta_{ij}\phi_j\right), \quad i = \overline{1, q},$$

$$\Delta y \approx \sum_{i=0}^{q} A_i \phi_i$$

При $q = 2$ получим метод Рунге-Кутты третьего порядка точности:

$$\begin{cases} A_0 + A_1 + A_2 = 1 \\ A_1\alpha_1 + A_2\alpha_2 = \frac{1}{2} \\ A_1\alpha_1^2 + A_2\alpha_2^2 = \frac{1}{3} \\ A_2\alpha_1\beta_{21} = \frac{1}{6} \\ \beta_{20} + \beta_{21} = \alpha_2 \\ \beta_{10} = \alpha_1 \end{cases}$$

$$\phi_0 = hf(x_n, y(x_n)); \quad \phi_1 = hf\left(x_n + \frac{h}{2}, y(x_n) + \frac{\phi_0}{2}\right)$$

$$\phi_2 = hf(x_n + h, y(x_n) - \phi_0 + 2\phi_1); \quad \Delta y \approx \frac{1}{6}(\phi_0 + 4\phi_1 + \phi_2)$$

А при $q = 1$ - второго порядка точности:

$$\begin{cases} A_0 + A_1 = 1 \\ A_1\alpha_1 = \frac{1}{2} \\ A_1\beta_{10} = \frac{1}{2} \end{cases}$$

$$\phi_{10} = hf_1(x_n, y(x_n), z(x_n)); \quad \phi_{20} = hf_2(x_n, y(x_n), z(x_n))$$

$$\phi_{11} = hf_1\left(x_n + \frac{2h}{3}, y(x_n) + \frac{2\phi_{10}}{3}, z(x_n) + \frac{2\phi_{20}}{3}\right); \quad \phi_{21} = hf_2\left(x_n + \frac{2h}{3}, y(x_n) + \frac{2\phi_{10}}{3}, z(x_n) + \frac{2\phi_{20}}{3}\right);$$

$$\Delta y \approx \frac{1}{4}(\phi_{10} + 3\phi_{11}); \quad \Delta z \approx \frac{1}{4}(\phi_{20} + 3\phi_{21})$$

2.6 Метод Адамса

Метод Адамса - многошаговый метод, поэтому искомая функция аппроксимируется используя значения в нескольких предыдущих узлах. Для экстраполяционного метода Адамса аппроксимация примет следующий вид:

$$y_{n+1} = y_n + h\sum_{i=0}^{q} A_i f(x_{n-i}, y_{n-i})$$

$$\begin{cases} \sum_{i=0}^{q} A_i = 1 \\ \sum_{i=0}^{q} A_i(-i)^j = \frac{1}{j+1}, \quad j = \overline{1, q} \end{cases}$$

При $q = 2$ получаем:

$$A_0 = \frac{23}{12}; \quad A_1 = -\frac{4}{3}; \quad A_2 = \frac{5}{12}$$

$$y_{n+1} = y_n + \frac{h}{12}(23f_n - 16f_{n-1} + 5f_{n-2})$$

Значения в первых трех точках находятся с помощью метода предиктор-корректор того же порядка точности.

Для интерполяционного метода Адамса аппроксимация примет следующий вид:

$$y_{n+1} = y_n + h\sum_{i=-1}^{q} A_i f(x_{n-i}, y_{n-i})$$

$$\begin{cases} \sum_{i=-1}^{q} A_i = 1 \\ \sum_{i=-1}^{q} A_i(-i)^j = \frac{1}{j+1}, \quad j = \overline{1, q+1} \end{cases}$$

При $q = 1$ получаем:

$$A_{-1} = \frac{5}{12}; \quad A_0 = \frac{3}{4}; \quad A_1 = -\frac{1}{12}$$

$$y_{n+1} = y_n + \frac{h}{12}(5(f_1)_{n+1} + 8(f_1)_n - (f_1)_{n-1})$$

$$z_{n+1} = z_n + \frac{h}{12}(5(f_2)_{n+1} + 8(f_2)_n - (f_2)_{n-1})$$

Полученная система решается с помощью МПИ. Итерации проводятся пока норма невязки изменения решения не окажется меньше ε. Значения в первых трех точках также находятся с помощью метода предиктор-корректор того же порядка точности

3. Результаты

Задача Коши для ОДУ-1:

$$y' = \frac{2xy^3}{1-x^2y^2}$$

$$x \subseteq [2,\ 2.45]$$

$$y(2) = 1$$

Задача Коши для системы ОДУ-1.

$$y' = yz + \frac{sin(x)}{1+x}$$

$$z' = z^2 + \frac{3.5x}{1+x^2}$$

$$x \subseteq [0,\ 1]$$

$$y(0) = 0$$

$$z(0) =- 0,4122$$

$$N = 9$$

Точные решения находились с помощью Wolfram Mathematica.

0	1	2	3	4	5	6	7	8	9
x									
2.00000000	2.05000000	2.10000000	2.15000000	2.20000000	2.25000000	2.30000000	2.35000000	2.40000000	2.45000000
y									
1.00000000	0.93537390	0.87448209	0.81681668	0.76186657	0.70908143	0.65780641	0.60714099	0.55555556	0.49937421
y_series									
0e+0	-1.29e-3	-2.18e-3	-2.69e-3	-2.8e-3	-2.45e-3	-1.52e-3	2.7e-4	3.61e-3	1.08e-2
y_exp_euler									
0e+0	-2.04e-3	-3.79e-3	-5.28e-3	-6.54e-3	-7.58e-3	-8.39e-3	-8.89e-3	-8.84e-3	-7e-3
y_imp_euler									
0e+0	1.92e-3	3.58e-3	5.01e-3	6.23e-3	7.25e-3	8.05e-3	8.56e-3	8.4e-3	4.9e-3
y_runge_kutta									
0e+0	-2.8e-3	-5.5e-3	-8.17e-3	-1.09e-2	-1.4e-2	-1.76e-2	-2.24e-2	-3.03e-2	-5.1e-2
y_pred_corr									
0e+0	-1.22e-7	-1.99e-7	-2.33e-7	-2.22e-7	-1.44e-7	7.92e-8	7.63e-7	3.56e-6	2.47e-5
y_ext_adams									
0e+0	-1.22e-7	-1.99e-7	-4.55e-6	1.3e-6	2.37e-5	7.9e-5	2.13e-4	5.84e-4	2.05e-3

0	1	2	3	4	5	6	7	8	9	10
x										
0.00000000	0.10000000	0.20000000	0.30000000	0.40000000	0.50000000	0.60000000	0.70000000	0.80000000	0.90000000	1.00000000
y										
0.00000000	0.00475249	0.01811130	0.03880250	0.06546980	0.09654140	0.13022300	0.16461400	0.19789700	0.22854500	0.25546400
z										
-0.41220000	-0.41200900	-0.37657800	-0.30629600	-0.20409800	0.07547780	0.07193670	0.22923600	0.38736400	0.53827500	0.67579400
y_series										
0e+0	-1.71e-5	-2.16e-5	-1.72e-5	-8.99e-6	-3.12e-6	-4.97e-6	-1.88e-5	-4.16e-5	-6.81e-5	-9.22e-5
0e+0	6.98e-5	1.72e-4	2.95e-4	4.17e-4	-1.5e-1	5.62e-4	5.4e-4	4.56e-4	3.29e-4	1.92e-4
y_exp_euler										
0e+0	-4.75e-3	-9.04e-3	-1.28e-2	-1.58e-2	-1.77e-2	-1.81e-2	-1.68e-2	-1.41e-2	-1.01e-2	-5.67e-3
0e+0	-1.72e-2	-3.64e-2	-5.64e-2	-7.54e-2	-2.42e-1	-1.01e-1	-1.04e-1	-9.96e-2	-8.81e-2	-7.18e-2
y_imp_euler										
0e+0	4.29e-3	7.3e-3	8.78e-3	8.72e-3	7.47e-3	5.72e-3	4.4e-3	4.46e-3	6.68e-3	1.15e-2
0e+0	4.46e-1	4.52e-1	4.25e-1	3.67e-1	1.3e-1	1.75e-1	5.63e-2	-6.57e-2	-1.83e-1	-2.9e-1
y_runge_kutta										
0e+0	-6.85e-5	-1.19e-4	-1.42e-4	-1.31e-4	-9.07e-5	-3.46e-5	1.26e-5	2.92e-5	2.86e-6	-6.44e-5
0e+0	-4.69e-4	-9.66e-4	-1.41e-3	-1.73e-3	-1.53e-1	-1.87e-3	-1.77e-3	-1.68e-3	-1.69e-3	-1.81e-3
y_pred_corr										
0e+0	3.17e-7	6.46e-7	1.01e-6	1.13e-6	7.85e-7	1.49e-7	-2.02e-6	-4e-6	-5.21e-6	-5.72e-6
0e+0	-4.75e-6	-6.89e-6	-7.55e-6	-6.73e-6	-1.51e-1	-5.78e-6	-9.45e-6	-1.4e-5	-1.91e-5	-2.05e-5
y_int_adams										
0e+0	3.17e-7	5.14e-6	8.16e-7	-7.73e-6	-1.41e-5	-1.23e-5	-6.44e-7	2.03e-5	4.52e-5	6.7e-5
0e+0	-4.75e-6	-1.02e-4	-2.21e-4	-3.43e-4	-1.51e-1	-4.91e-4	-4.76e-4	-3.99e-4	-2.84e-4	-1.6e-4

Ожидаемые погрешности методов:

Метод (ОДУ-1)	Погрешность метода	Метод (ОДУ-1 система)	Погрешность метода
Рядов	$O(h^4)$	Рядов	$O(h^4)$
Явный Эйлера	$O(h^1)$	Явный Эйлера	$O(h^1)$
Неявный Эйлера	$O(h^1)$	Неявный Эйлера	$O(h^1)$
Рунге-Кутта	$O(h^3)$	Рунге-Кутта	$O(h^2)$
Предиктор-корректор	$O(h^3)$	Предиктор-корректор	$O(h^3)$
Адамса	$O(h^4)$	Адамса	$O(h^3)$

4. Вывод

Как видно из сравнительной таблицы ожидаемых погрешностей, все полученные погрешности не выходят за пределы оценок. Однако не во всех методах сохраняется отношение реальной и ожидаемой погрешностей. Одной из причин этого является то, что погрешность оценена асимптотически, и настоящие константы для нее не учитываются. Для системы ОДУ-1 реальные погрешности относятся так же, как ожидаемые. Это говорит о том, что система ОДУ-1 устойчива. Наиболее точными оказались методы предиктор-корректор, Адамса, Рунге-Кутта и рядов. Все эти методы обладают порядком точности не менее 2. Тем не менее метод рядов имеет меньшее практическое применение по сравнению с остальными названными методами, так как для его реализации требуются производные высших порядков исходных функций, нахождение которых не всегда является возможным в случае работы с сеточными функциями.

4. Листинги

```
require('console.table')
const {eval, sum, subtract, factorial} = require('mathjs')
const {last, takeRight} = require('lodash')
const {sqrt, pow, abs} = Math

const array = n => [...Array(n).keys()]
const range = (a, b, n = 1) => array(n + 1).map(i => a + (b - a) / n * i)
const toFixed = n => v => Number(v.toPrecision(3)).toExponential()

const solution = x => eval(`(sqrt(25 - 4x^2) + 5) / (2x^2)`, {x})
const yd = [
    (x, y) => y,
    (x, y) => eval(`2 x y^3 / (1 - x^2 y^2)`, {x, y}),
    (x, y) => eval(`2 y^2 (-x^3 y^2 y' + x^2 y^3 + 3 x y' + y) / (1 - x^2 y^2)^2`, {x, y, "y'": yd[1](x, y)}),
    (x, y) => eval(`(4 y * (-x * (x^2 y^2 + 3) * yd ^2 - x * (x^2 y^2 + 3) * y^4 + (x^4 y^4 - 6 x^2 y^2 - 3) * y
yd ) - 2 x y^2 (x^4 y^4 - 4 x^2 y^2 + 3) ydd )/(x^2 y^2 - 1)^3`, {x, y, "yd": yd[1](x, y), "ydd": yd[2](x, y)})
]

const f = yd[1]
const EPS = 1E-8
const v = range(2, 2.45, 9)
const y0 = 1
const y = v.map(solution)
const h = v[1] - v[0]

const taylor = yd => (x, y) => sum(yd.map((f, i) => f(x, y) * pow(h, i) / factorial(i)))

const newton = (xn, y, yn = taylor(yd.slice(0, 2))(xn - h, y)) => {
    const fy = yn => eval(`(6 x y^2 - 10 x^3 y^4) / (1 - x^2 y^2)^2`, {x: xn, y: yn})
    const phi = yn => ((y + h * f(xn, yn) - h * yn * fy(yn)) / (1 - h * fy(yn)))
    while (abs(yn - (yn = phi(yn))) > EPS);
    return yn
}

const next_series = ({xn, yn}) => taylor(yd)(xn, yn);
const next_exp_euler = ({xn, yn}) => taylor(yd.slice(0, 2))(xn, yn);
const next_imp_euler = ({xn, yn}) => newton(xn + h, yn)
const next_runge_kutta = ({xn, yn}) => yn + h * f(xn - h / 2, yn + h / 2 * f(xn - h, yn))
const next_pred_corr = ({xn, yn}) => {
    const yn1_4 = yn + h / 4 * f(xn, yn)
    const yn1_2 = yn + h / 2 * f(xn + h / 4, yn1_4)
    const yn1 = yn + h * f(xn + h / 2, yn1_2)
    return yn + h / 6 * (f(xn, yn) + 4 * f(xn + h / 2, yn1_2) + f(xn + h, yn1))
}

const run = next => v.slice(1).reduce((yv, x) => {
    const xn = x - h
    const yn = last(yv)
    return [...yv, next({xn, yn})]
}, [y0])

const y_series = run(next_series)
const y_exp_euler = run(next_exp_euler)
const y_imp_euler = run(next_imp_euler)
const y_runge_kutta = run(next_runge_kutta)
const y_pred_corr = run(next_pred_corr)

const y_ext_adams = v.slice(3).reduce((yv, x) => {
    const ys = takeRight(yv, 3), xs = [x - 3 * h, x - 2 * h, x - h]
    return [...yv, ys[2] + h / 12 * (23 * f(xs[2], ys[2]) - 16 * f(xs[1], ys[1]) + 5 * f(xs[0], ys[0]))]
}, y_pred_corr.slice(0, 3))

const toAnswer = obj => {
    const key = Object.keys(obj)[0]
    return [[key], subtract(obj[key], y).map(toFixed(8))]
}

console.table([
    ['x'], v.map(v => v.toFixed(8)),
    ['y'], y.map(v => v.toFixed(8)),
    ...toAnswer({y_series}),
```

```
    ...toAnswer({y_exp_euler}),
    ...toAnswer({y_imp_euler}),
    ...toAnswer({y_runge_kutta}),
    ...toAnswer({y_pred_corr}),
    ...toAnswer({y_ext_adams})
])

require('console.table')
const {eval, sum, subtract, factorial} = require('mathjs')
const {last, takeRight} = require('lodash')
const {sqrt, pow, abs, max} = Math

const array = n => [...Array(n).keys()]
const range = (a, b, n = 1) => array(n + 1).map(i => a + (b - a) / n * i)
const toFixed = n => v => Number(v.toPrecision(3)).toExponential()

let zd = [], yd = []

const solutiony = x => {
    if (x < 0.05) return 0
    else if (x < 0.15) return 0.00475249
    else if (x < 0.25) return 0.0181113
    else if (x < 0.35) return 0.0388025
    else if (x < 0.45) return 0.0654698
    else if (x < 0.55) return 0.0965414
    else if (x < 0.65) return 0.130223
    else if (x < 0.75) return 0.164614
    else if (x < 0.85) return 0.197897
    else if (x < 0.95) return 0.228545
    else if (x < 1.05) return 0.255464
}
const solutionz = x => {
    if (x < 0.05) return -0.4122
    else if (x < 0.15) return -0.412009
    else if (x < 0.25) return -0.376578
    else if (x < 0.35) return -0.306296
    else if (x < 0.45) return -0.204098
    else if (x < 0.55) return 0.0754778
    else if (x < 0.65) return 0.0719367
    else if (x < 0.75) return 0.229236
    else if (x < 0.85) return 0.387364
    else if (x < 0.95) return 0.538275
    else if (x < 1.05) return 0.675794
}

yd = [
    (x, y, z) => y,
    (x, y, z) => eval(`-y z + sin(x) / (1 + x)`, {x, y, z}),
    (x, y, z) => eval(`-yd z - y zd + cos(x) / (1 + x) - sin(x) / (1 + x)^2`, {x, y, z, yd: yd[1](x, y, z), zd: zd[1](x, y, z)}),
    (x, y, z) => eval(`-ydd z - 2 yd zd - y zdd + -sin(x) / (1 + x) - 2 cos(x) / (1 + x)^2 + 2 sin(x) / (1 + x)^3`, {x, y, z, yd: yd[1](x, y, z), ydd: yd[2](x, y, z), zd: zd[1](x, y, z), zdd: zd[2](x, y, z)})
]

zd = [
    (x, y, z) => z,
    (x, y, z) => eval(`-z^2 + 3.5 x / (1 + x^2)`, {x, z}),
    (x, y, z) => eval(`-2 z zd + 3.5 / (1 + x^2) - 7x^2 / (1 + x^2)^2`, {x, z, zd: zd[1](x, y, z)}),
    (x, y, z) => eval(`-2 z zdd - 2 zd ^ 2 - 7x / (1 + x^2)^2 - 14x / (1 + x^2)^2 + 28 x^3 / (1 + x^2)^3`, {x, z, zd: zd[1](x, y, z), zdd: zd[2](x, y, z)})
]

const f1 = yd[1]
const f2 = zd[1]
const EPS = 1E-8
const v = range(0, 1, 10)
const y0 = 0
const z0 = -0.4122
const h = v[1] - v[0]

const y = v.map(solutiony)
const z = v.map(solutionz)

const taylor = fd => (x, y, z) => sum(fd.map((f, i) => f(x, y, z) * pow(h, i) / factorial(i)))
```

```
const newton = (xn1, yn, zn) => {
    let yn1 = taylor(yd.slice(0, 2))(xn1 - h, yn, zn)
    let zn1 = taylor(zd.slice(0, 2))(xn1 - h, yn, zn)
    const f1y = (y, z) => -z
    const f2z = (y, z) => -2 * z

    const phi = (f, fd) => (yn1, zn1) => ((yn + h * f(xn1, yn1, zn1) - h * yn1 * fd(yn1, zn1)) / (1 - h *
fd(yn1, zn1)))
    const phi1 = phi(f1, f1y)
    const phi2 = phi(f2, f2z)
    let err = 10
    while (err > EPS) {
        let prev_yn1 = yn1, prev_zn1 = zn1
        yn1 = phi1(yn1, zn1)
        zn1 = phi2(yn1, zn1)
        err = max(abs(yn1 - prev_yn1), abs(zn1 - prev_zn1))
    }
    return [yn1, zn1]
}

const next_series = ({xn, yn, zn}) => [taylor(yd)(xn, yn, zn), taylor(zd)(xn, yn, zn)];
const next_exp_euler = ({xn, yn, zn}) => [
    taylor(yd.slice(0, 2))(xn, yn, zn),
    taylor(zd.slice(0, 2))(xn, yn, zn)
];
const next_imp_euler = ({xn, yn, zn}) => newton(xn + h, yn, zn)
const next_runge_kutta = ({xn, yn, zn}) => {
    yn1 = h * f1(xn, yn, zn)
    zn1 = h * f2(xn, yn, zn)
    yn2 = h * f1(xn + 2 / 3 * h, yn + 2 / 3 * yn1, zn + 2 / 3 * zn1)
    zn2 = h * f2(xn + 2 / 3 * h, yn + 2 / 3 * yn1, zn + 2 / 3 * zn1)
    return [
        yn + 1 / 4 * (yn1 + 3 * yn2),
        zn + 1 / 4 * (zn1 + 3 * zn2)
    ]
}
const next_pred_corr = ({xn, yn, zn}) => {
    const yn1_4 = yn + h / 4 * f1(xn, yn, zn)
    const zn1_4 = zn + h / 4 * f2(xn, yn, zn)
    const yn1_2 = yn + h / 2 * f1(xn + h / 4, yn1_4, zn1_4)
    const zn1_2 = zn + h / 2 * f2(xn + h / 4, yn1_4, zn1_4)
    const yn1 = yn + h * f1(xn + h / 2, yn1_2, zn1_2)
    const zn1 = zn + h * f2(xn + h / 2, yn1_2, zn1_2)
    return [
        yn + h / 6 * (f1(xn, yn, zn) + 4 * f1(xn + h / 2, yn1_2, zn1_2) + f1(xn + h, yn1, zn1)),
        zn + h / 6 * (f2(xn, yn, zn) + 4 * f2(xn + h / 2, yn1_2, zn1_2) + f2(xn + h, yn1, zn1)),
    ]
}

const run = next => v.slice(1).reduce((yv, x) => {
    const xn = x - h
    const [yn, zn] = last(yv)
    return [...yv, next({xn, yn, zn})]
}, [[y0, z0]])

const y_series = run(next_series)
const y_exp_euler = run(next_exp_euler)
const y_imp_euler = run(next_imp_euler)
const y_runge_kutta = run(next_runge_kutta)
const y_pred_corr = run(next_pred_corr)

const y_int_adams = v.slice(2).reduce((yv, x) => {
    const yzs = takeRight(yv, 2), xs = [x - 2 * h, x - h, x]
    let yn1 = taylor(yd.slice(0, 2))(xs[1], yzs[1][0], yzs[1][1])
    let zn1 = taylor(zd.slice(0, 2))(xs[1], yzs[1][0], yzs[1][1])

    let err = 10
    while (err > EPS) {
        let prev_yn1 = yn1, prev_zn1 = zn1
        yn1 = yzs[1][0] + h / 12 * (5 * f1(xs[2], yn1, zn1) + 8 * f1(xs[1], yzs[1][0], yzs[1][1]) - f1(xs[0],
yzs[0][0], yzs[0][1]))
        zn1 = yzs[1][1] + h / 12 * (5 * f2(xs[2], yn1, zn1) + 8 * f2(xs[1], yzs[1][0], yzs[1][1]) - f2(xs[0],
yzs[0][0], yzs[0][1]))
```

```
        err = max(abs(yn1 - prev_yn1), abs(zn1 - prev_zn1))
    }
    return [...yv, [yn1, zn1]]
}, y_pred_corr.slice(0, 2))

const toAnswer = obj => {
    const key = Object.keys(obj)[0]
    return [
        [key],
        subtract(obj[key].map(v => v[0]), y).map(toFixed(8)),
        subtract(obj[key].map(v => v[1]), z).map(toFixed(8))
    ]
}

console.table([
    ['x'], v.map(v => v.toFixed(8)),
    ['y'], y.map(v => v.toFixed(8)),
    ['z'], z.map(v => v.toFixed(8)),
    ...toAnswer({y_series}),
    ...toAnswer({y_exp_euler}),
    ...toAnswer({y_imp_euler}),
    ...toAnswer({y_runge_kutta}),
    ...toAnswer({y_pred_corr}),
    ...toAnswer({y_int_adams})
])
```