Московский авиационный институт
(национальный исследовательский университет)

Институт № 8 «Компьютерные науки и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 4 по курсу «Численные методы»

Студент:	Н. О. Тимофеева
Преподаватель:	Д. Е. Пивоваров
Группа:	М8О-308Б-19
Вариант:	19
Дата:	
Оценка:	

Москва, 2022

Лабораторная работа 4

Методы решения начальных и краевых задач для обыкновенных дифференциальных уравнений (ОДУ) и систем ОДУ

4.1

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Исходный код

```
#include <iostream>
#include <vector>
#include <cmath>
#include <functional>
#include <tuple>

using namespace std;

const double EPS = 0.000000001;
void printData(const vector<tuple<double, double, double>> & v) {
    cout << "x = [";
    for (int i = 0; i < v.size(); ++i) {
        if (i) {
            cout << ", ";
        }
        cout << get<0>(v[i]);
    }
    cout << "]\n";
    cout << "y = [";
    for (int i = 0; i < v.size(); ++i) {
        if (i) {
            cout << ", ";
        }
        cout << get<1>(v[i]);
    }
    cout << "]\n";
}
bool border(double a, double b) {
    return (a < b) or (abs(b - a) < EPS);
}
class euler {
private:
    double l, r, y0, z0;
    function<double(double, double, double)> f, g;
public:
    euler(const double _l, const double _r,
        const function<double(double, double, double)> _f, const function<double(double, double, double)>
            _g,
        const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
    vector<tuple<double, double, double>> solve(double h) {
        double xi = l;
```

```
        double yi = y0;
        double zi = z0;
        vector<tuple<double, double, double>> res;
        res.push_back(make_tuple(xi, yi, zi));
        while (border(xi + h, r)) {
            double dy = h * f(xi, yi, zi);
            double dz = h * g(xi, yi, zi);
            xi += h;
            yi += dy;
            zi += dz;
            res.push_back(make_tuple(xi, yi, zi));
        }
        return res;
    }
};
class rungeKutta {
private:
    double l, r, y0, z0;
    function<double(double, double, double)> f, g;
public:
    rungeKutta(const double _l, const double _r,
        const function<double(double, double, double)> _f, const function<double(double, double, double)>
            _g,
        const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
    vector<tuple<double, double, double>> solve(double h) {
        double xi = l;
        double yi = y0;
        double zi = z0;
        vector<tuple<double, double, double>> res;
        res.push_back(make_tuple(xi, yi, zi));
        while (border(xi + h, r)) {
            double k1 = h * f(xi, yi, zi);
            double l1 = h * g(xi, yi, zi);
            double k2 = h * f(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
            double l2 = h * g(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
            double k3 = h * f(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
            double l3 = h * g(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
            double k4 = h * f(xi + h, yi + k3, zi + l3);
            double l4 = h * g(xi + h, yi + k3, zi + l3);
            double dy = (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
            double dz = (l1 + 2.0 * l2 + 2.0 * l3 + l4) / 6.0;
            xi += h;
            yi += dy;
            zi += dz;
            res.push_back(make_tuple(xi, yi, zi));
        }
        return res;
    }
};
class adams {
private:
    double l, r, y0, z0;
    function<double(double, double, double)> f, g;
public:
    adams(const double _l, const double _r,
        const function<double(double, double, double)> _f, const function<double(double, double, double)>
            _g,
        const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
    double calc_tuple(function<double(double, double, double)> f, tuple<double, double, double> xyz) {
        return f(get<0>(xyz), get<1>(xyz), get<2>(xyz));
    }
    vector<tuple<double, double, double>> solve(double h) {
```

```
        if (l + 3.0 * h > r) {
            throw invalid_argument("  ");
        }
        rungeKutta firstPoints(l, l + 3.0 * h, f, g, y0, z0);
        vector<tuple<double, double, double>> res = firstPoints.solve(h);
        size_t cnt = res.size();
        double xk = get<0>(res.back());
        double yk = get<1>(res.back());
        double zk = get<2>(res.back());
        while (border(xk + h, r)) {
            double dy = (h / 24.0) * (55.0 * calc_tuple(f, res[cnt - 1]) - 59.0 * calc_tuple(f, res[cnt - 2]) + 37.0 * calc_tuple(f, res[cnt - 3]) - 9.0 * calc_tuple(f, res[cnt - 4]));
            double dz = (h / 24.0) * (55.0 * calc_tuple(g, res[cnt - 1]) - 59.0 * calc_tuple(g, res[cnt - 2]) + 37.0 * calc_tuple(g, res[cnt - 3]) - 9.0 * calc_tuple(g, res[cnt - 4]));
            double xk1 = xk + h;
            double yk1 = yk + dy;
            double zk1 = zk + dz;
            res.push_back(make_tuple(xk1, yk1, zk1));
            ++cnt;
            dy = (h / 24.0) * (9.0 * calc_tuple(f, res[cnt - 1]) + 19.0 * calc_tuple(f, res[cnt - 2]) - 5.0 * calc_tuple(f, res[cnt - 3])
 + 1.0 * calc_tuple(f, res[cnt - 4]));
            dz = (h / 24.0) * (9.0 * calc_tuple(g, res[cnt - 1]) + 19.0 * calc_tuple(g, res[cnt - 2]) - 5.0 * calc_tuple(g, res[cnt - 3])
 + 1.0 * calc_tuple(g, res[cnt - 4]));
            xk += h;
            yk += dy;
            zk += dz;
            res.pop_back();
            res.push_back(make_tuple(xk, yk, zk));
        }
        return res;
    }
};
double rungeRomberg(const vector<tuple<double, double, double>> & y_2h, const vector<tuple<double, double, double>> & y_h, double p) {
    double coef = 1.0 / (pow(2, p) - 1.0);
    double res = 0.0;
    for (int i = 0; i < y_2h.size(); ++i) {
        res = max(res, coef * abs(get<1>(y_2h[i]) - get<1>(y_h[2 * i])));
    }
    return res;
}
double f(double x, double y, double z) {
    return z;
}
double g(double x, double y, double z) {
    return (12*y/(x*x));
}
int main() {
    cout.precision(5);
    cout << fixed;
    double l, r, y0, z0, h;
    cin >> l >> r;
    cin >> y0 >> z0 >> h;
    euler mEuler(l, r, f, g, y0, z0);
    vector<tuple<double, double, double>> solEuler = mEuler.solve(h);
    cout << " :" << endl;
    printData(solEuler);
    double eulerError = rungeRomberg(mEuler.solve(h), mEuler.solve(h / 2), 1);
    cout << " = " << eulerError << "\n\n";
    rungeKutta mRunge(l, r, f, g, y0, z0);
```

```cpp
    vector<tuple<double, double, double>> solRunge = mRunge.solve(h);
    cout << " -:" << endl;
    printData(solRunge);
    double rungeError = rungeRomberg(mRunge.solve(h), mRunge.solve(h / 2), 4);
    cout << " = " << rungeError << "\n\n";
    adams mAdams(l, r, f, g, y0, z0);
    vector<tuple<double, double, double>> solAdams = mAdams.solve(h);
    cout << " :" << endl;
    printData(solAdams);
    double adamsError = rungeRomberg(mAdams.solve(h), mAdams.solve(h / 2), 4);
    cout << " = " << adamsError << endl;
}
```

Входные данные

```
test:
1 2
2 1 0.1
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab4/lab4-1$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/Lab4/lab4-1$ ./a.out <test
Метод Эйлера:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.10000,2.44000,2.98826,3.73986,4.70364,5.89640,7.34001,9.06002,11.08480,13.44514]
Погрешность  = 1.18886

Метод Рунге-Кутты:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.21546,2.65236,3.31131,4.20604,5.35876,6.79765,8.55548,10.66883,13.17756,16.12455]
Погрешность = 0.00003

Метод Адамса:
x = [1.00000,1.10000,1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [2.00000,2.21546,2.65236,3.31131,4.20536,5.35786,6.79673,8.55461,10.66802,13.17683,16.12390]
Погрешность = 0.00007
```

4.2

Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

Исходный код

```
#include <iostream>
#include <cmath>
#include <exception>
#include <vector>
#include <tuple>
#include <functional>

using namespace std;

const double EPS = 0.000000001;
template<class T>
class tridiag_t {
private:
    int n;
    std::vector<T> a;
    std::vector<T> b;
    std::vector<T> c;
public:
    tridiag_t(const int & _n) : n(_n), a(n), b(n), c(n) {}
    tridiag_t(const std::vector<T> & _a, const std::vector<T> & _b, const std::vector<T> & _c) {
        if (!(_a.size() == _b.size() and _a.size() == _c.size())) {
            throw std::invalid_argument("Sizes of a, b, c are invalid");
        }
        n = _a.size();
        a = _a;
        b = _b;
        c = _c;
    }
    std::vector<T> solve(const std::vector<T> & d) {
        int m = d.size();
        if (n != m) {
            throw std::invalid_argument("Size of vector d is invalid");
        }
        std::vector<T> p(n);
        p[0] = -c[0] / b[0];
        std::vector<T> q(n);
        q[0] = d[0] / b[0];
        for (int i = 1; i < n; ++i) {
            p[i] = -c[i] / (b[i] + a[i] * p[i - 1]);
            q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]);
        }
        std::vector<T> x(n);
        x.back() = q.back();
        for (int i = n - 2; i >= 0; --i) {
            x[i] = p[i] * x[i + 1] + q[i];
        }
        return x;
    }
```

```
    friend std::istream & operator >> (std::istream & in, tridiag_t<T> & tridiag) {
        in >> tridiag.b[0] >> tridiag.c[0];
        for (int i = 1; i < tridiag.n - 1; ++i) {
            in >> tridiag.a[i] >> tridiag.b[i] >> tridiag.c[i];
        }
        in >> tridiag.a.back() >> tridiag.b.back();
        return in;
    }
    ~tridiag_t() = default;
};
void printData(const std::vector<std::tuple<double, double, double>> & v) {
    std::cout << "x = [";
    for (int i = 0; i < v.size(); ++i) {
        if (i) {
            std::cout << ", ";
        }
        std::cout << std::get<0>(v[i]);
    }
    std::cout << "]\n";
    std::cout << "y = [";
    for (int i = 0; i < v.size(); ++i) {
        if (i) {
            std::cout << ", ";
        }
        std::cout << std::get<1>(v[i]);
    }
    std::cout << "]\n";
}
bool border(double a, double b) {
    return (a < b) or (std::abs(b - a) < EPS);
}
class rungeKutta {
private:
    double l, r, y0, z0;
    std::function<double(double, double, double)> f, g;
public:
    rungeKutta(const double _l, const double _r,
        const std::function<double(double, double, double)> _f, const std::function<double(double, double, double)> _g,
        const double _y0, const double _z0) : l(_l), r(_r), f(_f), g(_g), y0(_y0), z0(_z0) {}
    std::vector<std::tuple<double, double, double>> solve(double h) {
        double xi = l;
        double yi = y0;
        double zi = z0;
        std::vector<std::tuple<double, double, double>> res;
        res.push_back(std::make_tuple(xi, yi, zi));
        while (border(xi + h, r)) {
            double k1 = h * f(xi, yi, zi);
            double l1 = h * g(xi, yi, zi);
            double k2 = h * f(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
            double l2 = h * g(xi + 0.5 * h, yi + 0.5 * k1, zi + 0.5 * l1);
            double k3 = h * f(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
            double l3 = h * g(xi + 0.5 * h, yi + 0.5 * k2, zi + 0.5 * l2);
            double k4 = h * f(xi + h, yi + k3, zi + l3);
            double l4 = h * g(xi + h, yi + k3, zi + l3);
            double dy = (k1 + 2.0 * k2 + 2.0 * k3 + k4) / 6.0;
            double dz = (l1 + 2.0 * l2 + 2.0 * l3 + l4) / 6.0;
            xi += h;
            yi += dy;
            zi += dz;
            res.push_back(std::make_tuple(xi, yi, zi));
        }
```

```
        return res;
    }
};
double rungeRomberg(const std::vector<std::tuple<double, double, double>> & y_2h, const std::vector<std::
     tuple<double, double, double>> & y_h, double p) {
    double coef = 1.0 / (std::pow(2, p) - 1.0);
    double res = 0.0;
    for (int i = 0; i < y_2h.size(); ++i) {
        res = std::max(res, coef * std::abs(std::get<1>(y_2h[i]) - std::get<1>(y_h[2 * i])));
    }
    return res;
}
class shooting {
private:
    double a, b;
    std::function<double(double, double, double)> f, g;
    double alpha, beta, y0;
    double delta, gamma, y1;

public:
    shooting(const double _a, const double _b,
             const std::function<double(double, double, double)> _f,
             const std::function<double(double, double, double)> _g,
             const double _alpha, const double _beta, const double _y0,
             const double _delta, const double _gamma, const double _y1)
            : a(_a), b(_b), f(_f), g(_g),
              alpha(_alpha), beta(_beta), y0(_y0),
              delta(_delta), gamma(_gamma), y1(_y1) {}

    double get_start_cond(double eta) {
        return (y0 - alpha * eta) / beta;
    }

    double get_eta_next(double eta_prev, double eta, const std::vector<std::tuple<double, double, double>>
         sol_prev, const std::vector<std::tuple<double, double, double>> sol) {
        double yb_prev = std::get<1>(sol_prev.back());
        double zb_prev = std::get<2>(sol_prev.back());
        double phi_prev = delta * yb_prev + gamma * zb_prev - y1;
        double yb = std::get<1>(sol.back());
        double zb = std::get<2>(sol.back());
        double phi = delta * yb + gamma * zb - y1;
        return eta - (eta - eta_prev) / (phi - phi_prev) * phi;
    }
    std::vector<std::tuple<double, double, double>> solve(double h, double eps) {
        double eta_prev = 1.0;
        double eta = 0.8;
        while (1) {
            double rungeKutta_z0_prev = get_start_cond(eta_prev);
            rungeKutta de_solver_prev(a, b, f, g, eta_prev, rungeKutta_z0_prev);
            std::vector<std::tuple<double, double, double>> sol_prev = de_solver_prev.solve(h);

            double rungeKutta_z0 = get_start_cond(eta);
            rungeKutta de_solver(a, b, f, g, eta, rungeKutta_z0);
            std::vector<std::tuple<double, double, double>> sol = de_solver.solve(h);

            double eta_next = get_eta_next(eta_prev, eta, sol_prev, sol);
            if (std::abs(eta_next - eta) < eps) {
                return sol;
            } else {
                eta_prev = eta;
                eta = eta_next;
            }
```

```
        }
    }
};
class finDif {
private:
    using fx = std::function<double(double)>;
    using tridiag = tridiag_t<double>;

    double a, b;
    fx p, q, f;
    double alpha, beta, y0;
    double delta, gamma, y1;

public:
    finDif(const double _a, const double _b,
        const fx _p, const fx _q, const fx _f,
        const double _alpha, const double _beta, const double _y0,
        const double _delta, const double _gamma, const double _y1)
        : a(_a), b(_b), p(_p), q(_q), f(_f),
        alpha(_alpha), beta(_beta), y0(_y0),
        delta(_delta), gamma(_gamma), y1(_y1) {}

    std::vector<std::tuple<double, double, double>> solve(double h) {
        int n = (b - a) / h;
        std::vector<double> xk(n + 1);
        for (int i = 0; i <= n; ++i) {
            xk[i] = a + h * i;
        }
        std::vector<double> a(n + 1);
        std::vector<double> b(n + 1);
        std::vector<double> c(n + 1);
        std::vector<double> d(n + 1);
        b[0] = h * alpha - beta;
        c[0] = beta;
        d[0] = h * y0;
        a.back() = -gamma;
        b.back() = h * delta + gamma;
        d.back() = h * y1;
        for (int i = 1; i < n; ++i) {
            a[i] = 1.0 - p(xk[i]) * h * 0.5;
            b[i] = -2.0 + h * h * q(xk[i]);
            c[i] = 1.0 + p(xk[i]) * h * 0.5;
            d[i] = h * h * f(xk[i]);
        }
        tridiag sys_eq(a, b, c);
        std::vector<double> yk = sys_eq.solve(d);
        std::vector<std::tuple<double, double, double>> res;
        for (int i = 0; i <= n; ++i) {
            res.push_back(std::make_tuple(xk[i], yk[i], NAN));
        }
        return res;
    }
};
double f(double x, double y, double z) {
    return z;
}
double fx(double x) {
    return 0.0;
}
double g(double x, double y, double z) {
    return ((-4 * x) * z - (4 * x * x + 2) * y);
}
```

```
double px(double x) {
    return 4 * x;
}
double qx(double x) {
    return 4 * x * x + 2;
}
int main() {
    cout.precision(5);
    cout << fixed;
    double h, eps;
    cin >> h >> eps;
    /*
      3
    alpha * y(a) + beta * y'(a) = y0
    delta * y(b) + gamma * y'(b) = y1
    */
    double a = 0, b = 2;
    double alpha = 0, beta = 1, y0 = 1;
    double delta = 4, gamma = -1, y1 = 23 * exp(-4);
    shooting mShooting(a, b, f, g, alpha, beta, y0, delta, gamma, y1);
    vector<tuple<double, double, double>> shootingSol = mShooting.solve(h, eps);
    cout << " :" << endl;
    printData(shootingSol);
    double shootingError = rungeRomberg(mShooting.solve(h, eps), mShooting.solve(h / 2, eps), 4);
    cout << "  = " << shootingError << "\n\n";
    finDif mFinDif(a, b, px, qx, fx, alpha, beta, y0, delta, gamma, y1);
    vector<tuple<double, double, double>> finDifSol = mFinDif.solve(h);
    cout << "- :" << endl;
    printData(finDifSol);
    double finDifError = rungeRomberg(mFinDif.solve(h), mFinDif.solve(h / 2), 2);
    cout << "  = " << finDifError << "\n\n";
}
```

Входные данные

```
test:
5
0.1
0.0001
```

Консоль

```
natalya@natalya-Ideapad-Z570:~/NumMeth/m4-2$ g++ main.cpp
natalya@natalya-Ideapad-Z570:~/NumMeth/m4-2$ ./a.out <test
Метод стрельбы:
x = [0.00000,0.10000,0.20000,0.30000,0.40000,0.50000,0.60000,0.70000,0.80000,0.90000,1.00000,1.10000,
1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [1.00074,1.08979,1.15366,1.18879,1.19363,1.16878,1.11680,1.04192,0.94952,0.84556,0.73602,0.62642,
0.52140,0.42451,0.33814,0.26355,0.20103,0.15008,0.10967,0.07846,0.05495]
Погрешность вычислений = 0.00005
```

```
Конечно-разностный метод:
x = [0.00000,0.10000,0.20000,0.30000,0.40000,0.50000,0.60000,0.70000,0.80000,0.90000,1.00000,1.10000,
1.20000,1.30000,1.40000,1.50000,1.60000,1.70000,1.80000,1.90000,2.00000]
y = [0.69395,0.79395,0.87415,0.93002,0.95887,0.96000,0.93474,0.88619,0.81878,0.73777,0.64872,0.55692,
0.46700,0.38266,0.30653,0.24014,0.18407,0.13812,0.10151,0.07311,0.05165]
Погрешность вычислений = 0.04388
```