Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторные работы по курсу «Численные методы»

Студент:	Ерофеева Е.С.
Преподаватель:	Пивоваров Д.Е.
Группа:	М8О-303Б-21
Дата:	
Оценка:	
Подпись:	

Москва, 2024

4.1 Методы Эйлера, Рунге-Кутты и Адамса

1 Постановка задачи

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.

Вариант: 6

6	$y'' + 4xy' + (4x^2 + 2)y = 0$, $y(0) = 1$, $y'(0) = 1$, $x \in [0,1], h = 0.1$	$y = (1+x)e^{-x^2}$

Рис. 1: Входные данные

2 Результаты работы

```
Error estimation by the Runge-Rombert method:
For the explicit Euler method: 0.250705
For the Runge-Kutta method: 1.11243e-05
For the Adams method: 0.0276392
Comparison with the exact solution:
For the explicit Euler method: 0.238889
For the Runge-Kutta method: 1.11308e-05
For the Adams method: 0.125643
```

Рис. 2: Вывод программы в консоли

3 Исходный код

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

using namespace std;
using vect = vector<double>;


vect f(vect& x) {
    vect res;
    for (double val : x)
        res.push_back((1 + val) / exp(val*val));
    return res;
}


double ddf(double x, double f, double df) {
    return -4*x*df - (4*x*x + 2)*f;
}


double sqr_error(const vect& a, const vect& b) {
    double res = 0;
    for (int i = 0; i < a.size(); i++)
        res += pow(a[i] - b[i], 2);
    return sqrt(res);
}

vect Euler(function<double(double, double, double)> ddy, pair<double, double>& borders
    , double y0, double z0, double h) {
    double x = borders.first;
    int N = (borders.second - borders.first) / h + 1;
    vect y(N), z(N);

    y[0] = y0;
    z[0] = z0;

    for (int i = 0; i < N - 1; i++) {
        y[i + 1] = y[i] + h * z[i];
        z[i + 1] = z[i] + h * ddy(x, y[i], z[i]);
        x += h;
    }

    return y;
}
```

```


pair<vect, vect> RungeKutta(function<double(double, double, double)> ddy, pair<double,
     double>& borders, double y0, double z0, double h) {
    double x = borders.first;
    int N = (borders.second - borders.first) / h + 1;
    vect y(N), z(N);

    y[0] = y0;
    z[0] = z0;

    for (int i = 0; i < N - 1; i++) {
        double K1 = h * z[i];
        double L1 = h * ddy(x, y[i], z[i]);
        double K2 = h * (z[i] + 0.5 * L1);
        double L2 = h * ddy(x + 0.5 * h, y[i] + 0.5 * K1, z[i] + 0.5 * L1);
        double K3 = h * (z[i] + 0.5 * L2);
        double L3 = h * ddy(x + 0.5 * h, y[i] + 0.5 * K2, z[i] + 0.5 * L2);
        double K4 = h * (z[i] + L3);
        double L4 = h * ddy(x + h, y[i] + K3, z[i] + L3);
        double delta_y = (K1 + 2 * K2 + 2 * K3 + K4) / 6;
        double delta_z = (L1 + 2 * L2 + 2 * L3 + L4) / 6;
        y[i + 1] = y[i] + delta_y;
        z[i + 1] = z[i] + delta_z;
        x += h;
    }

    return { y, z };
}


vect Adams(function<double(double, double, double)> ddy, pair<double, double>& borders
    , vect calc_y, vect& calc_z, double h) {
    double x = borders.first;
    int N = (borders.second - borders.first) / h + 1;
    vect y, z;

    for (int i = 0; i < 4; i++) {
        y.push_back(calc_y[i]);
        z.push_back(calc_z[i]);
    }

    for (int i = 3; i < N - 1; i++) {
        double z_next = z[i] + (h / 24) * (55 * ddy(x, y[i], z[i]) - 59 * ddy(x - h, y[
            i - 1], z[i - 1]) + 37 * ddy(x - 2 * h, y[i - 2], z[i - 2]) - 9 * ddy(x - 3
             * h, y[i - 3], z[i - 3]));
        double y_next = y[i] + (h / 24) * (55 * z[i] - 59 * z[i - 1] + 37 * z[i - 2] -
            9 * z[i - 3]);
```

```
        double z_i = z[i] + (h / 24) * (9 * ddy(x + h, y_next, z_next) + 19 * ddy(x, y[
            i], z[i]) - 5 * ddy(x - h, y[i - 1], z[i - 1]) + 1 * ddy(x - 2 * h, y[i -
            2], z[i - 2]));
        z.push_back(z_i);
        double y_i = y[i] + (h / 24) * (9 * z_next + 19 * z[i] - 5 * z[i - 1] + 1 * z[i
             - 2]);
        y.push_back(y_i);
        x += h;
    }

    return y;
}


vect RungeRomberg(vect y1, vect y2, double h1, double h2, double p) {
    if (h1 > h2) {
        int k = h1 / h2;
        vect y(y1.size());
        for (int i = 0; i < y1.size(); i++)
            y[i] = y2[i * k] + (y2[i * k] - y1[i]) / (pow(k, p) - 1);
        return y;
    }
    else {
        int k = h2 / h1;
        vect y(y2.size());
        for (int i = 0; i < y2.size(); i++)
            y[i] = y1[i * k] + (y1[i * k] - y2[i]) / (pow(k, p) - 1);
        return y;
    }
}


int main() {
    pair<double, double> borders = { 0, 1 };
    double y0 = 1, z0 = 1, h = 0.1;
    double h2 = h / 2;
    vect x = { borders.first };
    double last_el = borders.first;
    while (last_el < borders.second) {
        last_el += h;
        x.push_back(last_el);
    }

    vect y = f(x);

    vect y1, y2, y3, z, z2, y1_2, y2_2, y3_2;
    y1 = Euler(ddf, borders, y0, z0, h);
    tie(y2, z) = RungeKutta(ddf, borders, y0, z0, h);
    y3 = Adams(ddf, borders, y2, z, h);
```

```
    y1_2 = Euler(ddf, borders, y0, z0, h2);
    tie(y2_2, z2) = RungeKutta(ddf, borders, y0, z0, h2);
    y3_2 = Adams(ddf, borders, y2_2, z2, h2);
    cout << "Error estimation by the Runge-Rombert method:" << endl;
    cout << "For the explicit Euler method: " << sqr_error(y1, RungeRomberg(y1, y1_2, h
        , h2, 1)) << endl;
    cout << "For the Runge-Kutta method: " << sqr_error(y2, RungeRomberg(y2, y2_2, h,
        h2, 4)) << endl;
    cout << "For the Adams method: " << sqr_error(y3, RungeRomberg(y3, y3_2, h, h2, 4))
         << endl;
    cout << "Comparison with the exact solution:" << endl;
    cout << "For the explicit Euler method: " << sqr_error(y1, y) << endl;
    cout << "For the Runge-Kutta method: " << sqr_error(y2, y) << endl;
    cout << "For the Adams method: " << sqr_error(y3, y) << endl;
}
```

4.2 Метод стрельбы и конечно-разностный метод

4 Постановка задачи

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.

Вариант: 6

6	$y''-2(1+(\tan x)^2)y=0,$ $y(0)=0,$ $y(\frac{\pi}{6})=-\frac{\sqrt{3}}{3}$	$y(x)=-\tan x$

Рис. 3: Входные данные

5 Результаты работы

```
Shooting method:
x: 0.000000000  y: -0.453482850  exact y: -0.000000000
x: 0.100000000  y: -0.458032826  exact y: -0.100334672
x: 0.200000000  y: -0.471867766  exact y: -0.202710036
x: 0.300000000  y: -0.495565945  exact y: -0.309336250
x: 0.400000000  y: -0.530173576  exact y: -0.422793219
x: 0.500000000  y: -0.577350269  exact y: -0.546302490
Runge-Rumberg error:
0.000000097

Finite-difference method:
x: 0.000000000  y: 0.000000000  exact y: -0.000000000
x: 0.100000000  y: -0.120237418  exact y: -0.100334672
x: 0.200000000  y: -0.239270443  exact y: -0.202710036
x: 0.300000000  y: -0.355906686  exact y: -0.309336250
x: 0.400000000  y: -0.468977633  exact y: -0.422793219
x: 0.500000000  y: -0.577350269  exact y: -0.546302490
Runge-Rumberg error:
0.000006303
```

Рис. 4: Вывод программы в консоли

6 Исходный код

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

using namespace std;
using tddd = tuple<double, double, double>;
using func = function<double(double, double, double)>;
using vect = vector<tddd>;
using vec = vector<double>;

const double PI = acos(-1.0);
const double EPS = 1e-9;

double exact_y(double x) {
    return -tan(x);
}

double g(double x, double y, double dy) {
    return 2 * y * (1+(tan(x))*(tan(x)));
}

double f(double x, double y, double dy) {
    (void)x;
    (void)y;
    return dy;
}

double px(double x) {
    return -tan(x);
}

double qx(double x) {
    return 2.0;
}

double fx(double x) {
    (void)x;
    return 0.0;
}

bool leq(double a, double b) {
```

```
    return (a < b) || (abs(b - a) < EPS);
}

template <class T>
class Trid {
private:
    const T EPS = 1e-6;

    int n;
    vector<T> a, b, c;
public:
    Trid(const int& size) : n(size), a(n), b(n), c(n) {}
    Trid(vector<double>& _a, vector<double>& _b, vector<double>& _c) : n(_a.size()), a(_a), b(_b), c(_c) {};

    vector<T> Solve(const vector<T>& d) {
        vector<T> p(n);
        p[0] = -c[0] / b[0];
        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]);
        }
        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 istream& operator >> (istream& in, Trid<T>& t) {
        in >> t.b[0] >> t.c[0];
        for (int i = 1; i < t.n - 1; ++i) {
            in >> t.a[i] >> t.b[i] >> t.c[i];
        }
        in >> t.a.back() >> t.b.back();
        return in;
    }

    ~Trid() = default;
};


vect runge_solve(double l, double r, double y0, double z0, double h) {
    vect res;
    double xk = l;
    double yk = y0;
```

```
    double zk = z0;
    res.push_back(make_tuple(xk, yk, zk));
    while (leq(xk + h, r)) {
        double K1 = h * f(xk, yk, zk);
        double L1 = h * g(xk, yk, zk);
        double K2 = h * f(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
        double L2 = h * g(xk + 0.5 * h, yk + 0.5 * K1, zk + 0.5 * L1);
        double K3 = h * f(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
        double L3 = h * g(xk + 0.5 * h, yk + 0.5 * K2, zk + 0.5 * L2);
        double K4 = h * f(xk + h, yk + K3, zk + L3);
        double L4 = h * g(xk + h, yk + K3, zk + 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;
        xk += h;
        yk += dy;
        zk += dz;
        res.push_back(make_tuple(xk, yk, zk));
    }
    return res;
}

double runge_romberg_max(const vect& y_2h, const vect& y_h, double p) {
    double coef = 1.0 / (pow(2, p) - 1.0);
    double res = 0.0;
    for (size_t 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 runge_romberg(double y1, double y2, int64_t p) {
    return abs((y1 - y2) / (pow(2, p) - 1));
}

class fin_dif {
private:

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

public:
    fin_dif(const double _a, const double _b,
        const func _p, const func _q, const func _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) {}


};

vect shooting_solve(double a, double b, double alpha, double beta, double y0, double
    delta, double gamma, double y1, double h, double eps) {
    double eta_prev = 0.9;
    double eta = 0.8;
    while (1) {
        vect sol_prev = runge_solve(a, b, eta_prev, y0, h),
            sol = runge_solve(a, b, eta, y0, h);

        double yb_prev = get<1>(sol_prev.back());
        double zb_prev = get<2>(sol_prev.back());
        double phi_prev = delta * yb_prev + gamma * zb_prev - y1;
        double yb = get<1>(sol.back());
        double zb = get<2>(sol.back());
        double phi = delta * yb + gamma * zb - y1;
        double eta_next = eta - (eta - eta_prev) / (phi - phi_prev) * phi;
        if (abs(eta_next - eta) < eps) {
            return sol;
        }
        else {
            eta_prev = eta;
            eta = eta_next;
        }
    }
}

using tridiag = Trid<double>;
vect fin_dif_solve(double a, double b,
    double alpha, double beta, double y0,
    double delta, double gamma, double y1, double h) {
    size_t n = (b - a) / h;
    vec xk(n + 1);
    for (size_t i = 0; i <= n; ++i) {
        xk[i] = a + h * i;
    }
    vec A(n + 1);
    vec B(n + 1);
    vec C(n + 1);
    vec D(n + 1);
    B[0] = (alpha - beta / h);
    C[0] = beta / h;
    D[0] = y0;
    A.back() = -gamma / h;
    B.back() = delta + gamma / h;
    D.back() = y1;
```

```
    for (size_t i = 1; i < n; ++i) {
        A[i] = 1.0 - px(xk[i]) * h * 0.5;
        B[i] = -2.0 + h * h * qx(xk[i]);
        C[i] = 1.0 + px(xk[i]) * h * 0.5;
        D[i] = h * h * fx(xk[i]);
    }
    tridiag sys_eq(A, B, C);
    vec yk = sys_eq.Solve(D);
    vect res;
    for (size_t i = 0; i <= n; ++i) {
        res.push_back(make_tuple(xk[i], yk[i], NAN));
    }
    return res;
}



int main() {
    cout.precision(6);
    cout << fixed;
    double h = 0.1, eps = 0.001;

    double a = 0, b = PI/6;
    double alpha = 1, beta = 0, y0 = 0;
    double delta = 1, gamma = 0, y1 = -sqrt(3)/3;
    cout << "Shooting method:" << endl;
    vector<tddd> sol_shooting_h1 = shooting_solve(a, b, alpha, beta, y0, delta, gamma,
        y1, h, eps),
        sol_shooting_h2 = shooting_solve(a, b, alpha, beta, y0, delta, gamma, y1, h /
            2, eps);
    for (int i = 0; i < sol_shooting_h1.size(); ++i) {
        double ex_y = exact_y(get<0>(sol_shooting_h1[i]));
        cout << fixed << setprecision(9) << "x: " << get<0>(sol_shooting_h1[i]) << " y:
             " << get<1>(sol_shooting_h1[i]) << " exact y: " << ex_y << endl;
    }
    cout << "Runge-Rumberg error:" << endl;
    double shooting_err = runge_romberg_max(sol_shooting_h1, sol_shooting_h2, 4);
    cout << shooting_err << endl << endl;

    vector<tddd> sol_fin_dif_h1 = fin_dif_solve(a, b, alpha, beta, y0, delta, gamma, y1
        , h),
        sol_fin_dif_h2 = fin_dif_solve(a, b, alpha, beta, y0, delta, gamma, y1, h / 2);
    cout << "Finite-difference method:" << endl;
    for (int i = 0; i < sol_fin_dif_h1.size(); ++i) {
        double ex_y = exact_y(get<0>(sol_fin_dif_h1[i]));
        cout << fixed << setprecision(9) << "x: " << get<0>(sol_fin_dif_h1[i]) << " y:
            " << get<1>(sol_fin_dif_h1[i]) << " exact y: " << ex_y << endl;
    }
    cout << "Runge-Rumberg error:" << endl;
```

```
    double fin_dif_err = runge_romberg_max(sol_fin_dif_h1, sol_fin_dif_h2, 2);
    cout << fin_dif_err << endl;
}
```