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:	Белов И.А.
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: 2

2	$y''+y-2\cos x=0$, $y(0)=1,$ $y'(0)=0,$ $x\in[0,1], h=0.1$	$y = x\sin x + \cos x$

Рис. 1: Входные данные

2 Результаты работы

Эйлер результаты:

x	y_numeric	y_exact	error
0.000000	1.000000	1.000000	0.000000
0.100000	1.000000	1.004988	0.004988
0.200000	1.010000	1.019800	0.009800
0.300000	1.029900	1.043993	0.014092
0.400000	1.059301	1.076828	0.017527
0.500000	1.097511	1.117295	0.019785
0.600000	1.143548	1.164121	0.020573
0.700000	1.196162	1.215795	0.019633
0.800000	1.253847	1.270592	0.016745
0.900000	1.314867	1.326604	0.011737
1.000000	1.377283	1.381773	0.004490

Рунге-Кутта результаты:

x	y_numeric	y_exact	error
0.000000	1.000000	1.000000	0.000000
0.100000	1.004988	1.004988	0.000000
0.200000	1.019800	1.019800	0.000000
0.300000	1.043992	1.043993	0.000000
0.400000	1.076828	1.076828	0.000000
0.500000	1.117295	1.117295	0.000000
0.600000	1.164121	1.164121	0.000000
0.700000	1.215794	1.215795	0.000001
0.800000	1.270591	1.270592	0.000001
0.900000	1.326603	1.326604	0.000001
1.000000	1.381772	1.381773	0.000001

Адамс-Башфорт-Мултон результаты:

x	y_numeric	y_exact	error
0.000000	1.000000	1.000000	0.000000
0.100000	1.004988	1.004988	0.000000
0.200000	1.019800	1.019800	0.000000
0.300000	1.043992	1.043993	0.000000
0.400000	1.077270	1.076828	0.000441
0.500000	1.117444	1.117295	0.000149
0.600000	1.163298	1.164121	0.000823
0.700000	1.213184	1.215795	0.002610
0.800000	1.265308	1.270592	0.005284
0.900000	1.317710	1.326604	0.008895
1.000000	1.368308	1.381773	0.013465

3 Исходный код

```
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

//
const double h = 0.1;
const int N = 10; //   (1/h)
const double x0 = 0.0;
const double y0_initial = 1.0; //
const double dy0_initial = 0.0;

//
double f(double x, double y, double dy) {
    return 2 * cos(x) - y;
}

//
std::vector<double> exact_solution(double x) {
    return {x * sin(x) + cos(x), sin(x) + x * cos(x)};
}

//
std::vector<double> euler_method() {
    std::vector<double> y(N + 1), dy(N + 1);
    y[0] = y0_initial;
    dy[0] = dy0_initial;
    for (int i = 0; i < N; ++i) {
        double x = x0 + i * h;
        y[i + 1] = y[i] + h * dy[i];
        dy[i + 1] = dy[i] + h * f(x, y[i], dy[i]);
    }
    return y;
}

//  - 4-
std::vector<double> runge_kutta_method() {
    std::vector<double> y(N + 1), dy(N + 1);
    y[0] = y0_initial;
    dy[0] = dy0_initial;
    for (int i = 0; i < N; ++i) {
        double x = x0 + i * h;
        double k1 = h * dy[i];
        double l1 = h * f(x, y[i], dy[i]);
        double k2 = h * (dy[i] + 0.5 * l1);
        double l2 = h * f(x + 0.5 * h, y[i] + 0.5 * k1, dy[i] + 0.5 * l1);
        double k3 = h * (dy[i] + 0.5 * l2);
```

```cpp
        double l3 = h * f(x + 0.5 * h, y[i] + 0.5 * k2, dy[i] + 0.5 * l2);
        double k4 = h * (dy[i] + l3);
        double l4 = h * f(x + h, y[i] + k3, dy[i] + l3);
        y[i + 1] = y[i] + (k1 + 2 * k2 + 2 * k3 + k4) / 6.0;
        dy[i + 1] = dy[i] + (l1 + 2 * l2 + 2 * l3 + l4) / 6.0;
    }
    return y;
}

//  --
std::vector<double> adams_bashforth_moulton_method() {
    std::vector<double> y = runge_kutta_method(); //   -
    std::vector<double> dy(N + 1);
    dy[0] = dy0_initial;
    for (int i = 0; i < N; ++i) {
        double x = x0 + i * h;
        dy[i + 1] = dy[i] + h * f(x, y[i], dy[i]);
    }
    for (int i = 3; i < N; ++i) {
        double x = x0 + i * h;
        y[i + 1] = y[i] + h * (55 * dy[i] - 59 * dy[i - 1] + 37 * dy[i - 2] - 9 * dy[i - 3]) / 24.0;
        double dy_pred = dy[i] + h * f(x + h, y[i + 1], dy[i + 1]);
        y[i + 1] = y[i] + h * (9 * dy_pred + 19 * dy[i] - 5 * dy[i - 1] + dy[i - 2]) / 24.0;
        dy[i + 1] = dy_pred;
    }
    return y;
}

//
void print_results(const std::vector<double>& y_numeric, const std::string& method) {
    std::cout << method << " :\n";
    std::cout << "x\t\ty_numeric\ty_exact\t\terror\n";
    for (int i = 0; i <= N; ++i) {
        double x = x0 + i * h;
        double y_exact = exact_solution(x)[0];
        std::cout << std::fixed << std::setprecision(6) << x << "\t" << y_numeric[i] << "\t" << y_exact << "\t" << fabs(y_exact - y_numeric[i]) << "\n";
    }
    std::cout << "\n";
}

int main() {
    //
    std::vector<double> y_euler = euler_method();
    //   -
    std::vector<double> y_runge_kutta = runge_kutta_method();
    //   --
```

```
    std::vector<double> y_adams = adams_bashforth_moulton_method();

    //
    print_results(y_euler, "");
    print_results(y_runge_kutta, "-");
    print_results(y_adams, "--");

    return 0;
}
```

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: 2

2	$xy''+2y'-xy=0,$ $y(1)=e^{-1},$ $y(2)=0{,}5e^{-2}$	$y(x)=\dfrac{e^{-x}}{x}$

Рис. 2: Входные данные

5 Результаты работы

```
Метод стрельбы результаты:
x               y_numeric       y_exact         error
1.000000        0.367879        0.367879        0.000000
1.100000        0.278552        0.302610        0.024058
1.200000        0.206666        0.250995        0.044329
1.300000        0.147748        0.209640        0.061891
1.400000        0.098620        0.176141        0.077520
1.500000        0.056963        0.148753        0.091790
1.600000        0.021048        0.126185        0.105137
1.700000        -0.010444       0.107461        0.117905
1.800000        -0.038536       0.091833        0.130368
1.900000        -0.064036       0.078720        0.142756
2.000000        -0.087595       0.067668        0.155262

Конечно-разностный метод результаты:
x               y_numeric       y_exact         error
1.000000        -nan    0.367879        nan
1.100000        -nan    0.302610        nan
1.200000        -nan    0.250995        nan
1.300000        -nan    0.209640        nan
1.400000        -nan    0.176141        nan
1.500000        -nan    0.148753        nan
1.600000        -nan    0.126185        nan
1.700000        -nan    0.107461        nan
1.800000        -nan    0.091833        nan
1.900000        -nan    0.078720        nan
2.000000        -nan    0.067668        nan
```

Рис. 3: Вывод программы

6 Исходный код

```cpp
#include <iostream>
#include <vector>
#include <cmath>
#include <iomanip>

const double h = 0.1;
const int N = (2 - 1) / h; //    (1/h)
const double x1 = 1.0;
```

```
const double x2 = 2.0;
const double y_initial = exp(-1); //
const double y_final = 0.5 * exp(-2);

double exact_solution(double x) {
    return exp(-x) / x;
}

//
double f1(double x, double y, double dy) {
    return dy;
}

double f2(double x, double y, double dy) {
    return (x * y - 2 * dy) / x;
}

std::vector<double> shooting_method(double alpha) {
    std::vector<double> y(N + 1), dy(N + 1);
    y[0] = y_initial;
    dy[0] = alpha;
    for (int i = 0; i < N; ++i) {
        double x = x1 + i * h;
        double k1_y = h * f1(x, y[i], dy[i]);
        double k1_dy = h * f2(x, y[i], dy[i]);
        double k2_y = h * f1(x + 0.5 * h, y[i] + 0.5 * k1_y, dy[i] + 0.5 * k1_dy);
        double k2_dy = h * f2(x + 0.5 * h, y[i] + 0.5 * k1_y, dy[i] + 0.5 * k1_dy);
        double k3_y = h * f1(x + 0.5 * h, y[i] + 0.5 * k2_y, dy[i] + 0.5 * k2_dy);
        double k3_dy = h * f2(x + 0.5 * h, y[i] + 0.5 * k2_y, dy[i] + 0.5 * k2_dy);
        double k4_y = h * f1(x + h, y[i] + k3_y, dy[i] + k3_dy);
        double k4_dy = h * f2(x + h, y[i] + k3_y, dy[i] + k3_dy);
        y[i + 1] = y[i] + (k1_y + 2 * k2_y + 2 * k3_y + k4_y) / 6.0;
        dy[i + 1] = dy[i] + (k1_dy + 2 * k2_dy + 2 * k3_dy + k4_dy) / 6.0;
    }
    return y;
}

std::vector<double> finite_difference_method() {
    std::vector<double> a(N + 1, 0), b(N + 1, 0), c(N + 1, 0), d(N + 1, 0), y(N + 1, 0);
    double x;
    for (int i = 1; i < N; ++i) {
        x = x1 + i * h;
        a[i] = 1.0 - h / (2 * x);
        b[i] = -(2.0 + h * h);
        c[i] = 1.0 + h / (2 * x);
        d[i] = 0; // :
    }
    d[0] = y_initial;
```

```
    d[N] = y_final;

    //
    for (int i = 1; i <= N; ++i) {
        double m = a[i] / b[i - 1];
        b[i] = b[i] - m * c[i - 1];
        d[i] = d[i] - m * d[i - 1];
    }

    //
    y[N] = d[N] / b[N];
    for (int i = N - 1; i >= 0; --i) {
        y[i] = (d[i] - c[i] * y[i + 1]) / b[i];
    }
    return y;
}

void print_results(const std::vector<double>& y_numeric, const std::string& method) {
    std::cout << method << " :\n";
    std::cout << "x\t\ty_numeric\ty_exact\t\terror\n";
    for (int i = 0; i <= N; ++i) {
        double x = x1 + i * h;
        double y_exact = exact_solution(x);
        std::cout << std::fixed << std::setprecision(6) << x << "\t" << y_numeric[i] <<
             "\t" << y_exact << "\t" << fabs(y_exact - y_numeric[i]) << "\n";
    }
    std::cout << "\n";
}

int main() {
    //
    double alpha = -1.0; //    y'
    std::vector<double> y_shooting = shooting_method(alpha);

    // -
    std::vector<double> y_finite_difference = finite_difference_method();

    //
    print_results(y_shooting, " ");
    print_results(y_finite_difference, "- ");

    return 0;
}
```