МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационыые технологиии прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №4
по курсу «Численные методы»

Студент: Д. Д. Наумов
Преподаватель: И. Э. Иванов
Группа: М8О-406Б-17
Дата:
Оценка:
Подпись:

Москва, 2021

Часть 1

Задание

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки h. 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: 3

Задача Коши:

$$\begin{cases} y'' - 2y - 4x^2 e^{x^2} = 0 \\ y(0) = 3 \\ y'(0) = 0 \end{cases} \qquad x \in [0, 1], \quad h = 0.1$$

Точное решение:

$$y = e^{x^2} + e^{x\sqrt{2}} + e^{-x\sqrt{2}}$$

Теория

Рассматривается задача Коши для одного дифференциального уравнения первого порядка, разрешенного относительно производной

$$\begin{cases} y' = f(x) \\ y(x_0) = y_0 \end{cases}$$

Требуется найти решение на отрезке $[a, b]$, где $x_0 = a$
Введем разностную сетку на отрезке $[a, b]$:

$$\Omega = x_k = x_0 + kh, \qquad k = 0, 1, \ldots, N, \quad h = |b - a|/N$$

Формула метода Эйлера:

$$y_{k+1} = y_k + hf(x_k, y_k)$$

Все рассмотренные выше явные методы являются вариантами методов Рунге-Кутты. Семейство явных методов Рунге-Кутты p-го порядка записывается в виде совокупности формул:

$$y_{k+1} = y_k + \Delta y$$

$$\Delta y_i = \sum_{j=1}^{p} c_j K_j^i$$

$$K_i^k = hf\left(x_k + a_i h, y_k + h\sum_{j=1}^{i} b_{ij} K_j^k\right), \qquad i = 2, 3, \ldots, p$$

Метод Рунге-Кутты четвертого порядка точности

$$y_{k+1} = y_k + \Delta y$$

$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k)$$

$$K_1^k = hf\left(x_k, y_k\right)$$
$$K_2^k = hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k\right)$$
$$K_3^k = hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k\right)$$
$$K_4^k = hf\left(x_k + h, y_k + K_3^k\right)$$

Рассматривается задача Коши для системы дифференциальных уравнений первого порядка разрешенных относительно производной

$$\begin{cases} y_1' = f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' = f_2(x, y_1, y_2, \ldots, y_n) \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ y_n' = f_n(x, y_1, y_2, \ldots, y_n) \end{cases}$$

$$y_1(x_0) = y_{01}$$
$$y_2(x_0) = y_{02}$$
$$\ldots\ldots\ldots\ldots$$
$$y_n(x_0) = y_{0n}$$

Формулы метода Рунге-Кутты 4-го порядка точности для решения системы следующие

$$y_{k+1} = y_k + \Delta y$$
$$z_{k+1} = z_k + \Delta z$$

$$\Delta y_k = \frac{1}{6}(K_1^k + 2K_2^k + 2K_3^k + K_4^k)$$
$$\Delta z_k = \frac{1}{6}(L_1^k + 2L_2^k + 2L_3^k + L_4^k)$$

$$
\begin{aligned}
K_1^k &= hf\left(x_k, y_k, z_k\right) \\
L_1^k &= hg\left(x_k, y_k, z_k\right) \\
K_2^k &= hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k, z_k + \frac{1}{2}L_1^k\right) \\
L_2^k &= hg\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_1^k, z_k + \frac{1}{2}L_1^k\right) \\
K_3^k &= hf\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k, z_k + \frac{1}{2}L_2^k\right) \\
L_3^k &= hg\left(x_k + \frac{1}{2}h, y_k + \frac{1}{2}K_2^k, z_k + \frac{1}{2}L_2^k\right) \\
K_4^k &= hf\left(x_k + h, y_k + K_3^k, z_k + L_3^k\right) \\
L_4^k &= hg\left(x_k + h, y_k + K_3^k, z_k + L_3^k\right)
\end{aligned}
$$

При использовании интерполяционного многочлена 3-ей степени построенного по значениям подынтегральной функции в последних четырех узлах получим метод Адамса четвертого порядка точности:

$$y_{k+1} = y_k + \frac{h}{24}\left(55f_k - 59f_{k-1} + 37f_{k-2} - 9f_{k-3}\right)$$

Метод Адамса как и все многошаговые методы не является самостартующим, то есть для того, что бы использовать метод Адамса необходимо иметь решения в первых четырех узлах. В узле решение известно из начальных условий, а в других трех узлах решения можно получить с помощью подходящего одношагового метода, например: метода Рунге-Кутты четвертого порядка.

Реализация

Метод Эйлера

```
class EilerMethod2 {
public:
    EilerMethod2(Function3* f, double left, double right, double y0, double z0) :
        left(left),
        right(right),
        f(f),
        y0(y0),
        z0(z0) {
    }

    void operator() (double h) {
        x = GetValInRange(left, right, h);
        y.clear();
        z.clear();
        y.push_back(y0);
        z.push_back(z0);

        int n = x.size() - 1;
        for (int i = 0; i < n; ++i) {
            double z_next = z[i] + h * (*f)(x[i], y[i], z[i]);
            z.push_back(z_next);
            double y_next = y[i] + h * z[i];
            y.push_back(y_next);
        }
    }

    vdouble GetY() {
        return y;
    }

    vdouble GetZ() {
        return z;
    }


private:
    double left;
    double right;
    Function3* f;
    double h;
    double y0;
    double z0;

    vdouble x;
    vdouble y;
```

```
46          vdouble z;
47  };
```

Метод Рунге-Кутты

```
1       class RungeKutta2 {
2       public:
3           RungeKutta2(CauchyProblem t) :
4               left(t.left),
5               right(t.right),
6               f(t.f),
7               y0(t.y0),
8               z0(t.z0) {
9           }
10
11          vdouble operator() (double h) {
12              x = GetValInRange(left, right, h);
13              int n = x.size();
14              y.clear();
15              z.clear();
16              y.push_back(y0);
17              z.push_back(z0);
18
19              for (int i = 0; i < n - 1; ++i) {
20                  vdouble k(ORDER);
21                  vdouble l(ORDER);
22                  for (int j = 0; j < ORDER; ++j) {
23                      if (j == 0) {
24                          l[j] = h * (*f)(x[i], y[i], z[i]);
25                          k[j] = h * z[i];
26                      }
27                      else if (j == 3) {
28                          l[j] = h * (*f)(x[i] + h, y[i] + k[j - 1], z[i] + l[j - 1]);
29                          k[j] = h * (z[i] + l[j - 1]);
30                      } else {
31                          l[j] = h * (*f)(x[i] + 0.5*h, y[i] + 0.5*k[j - 1], z[i] + 0.5*l[j
                                - 1]);
32                          k[j] = h * (z[i] + 0.5*l[j - 1]);
33                      }
34                  }
35
36                  double y_next = y[i] + dif(k);
37                  double z_next = z[i] + dif(l);
38                  y.push_back(y_next);
39                  z.push_back(z_next);
40              }
41
42              return y;
43          }
```

```
44
45          vdouble GetY() {
46              return y;
47          }
48
49          vdouble GetZ() {
50              return z;
51          }
52
53          vdouble Get() {
54              return y;
55          }
56
57          Function3* f;
58          double left;
59          double right;
60          double y0;
61          double z0;
62
63      private:
64          double dif (const vdouble& k) {
65              return (k[0] + 2*(k[1] + k[2]) + k[3]) / 6;
66          }
67
68          double g (double z) {
69              return z;
70          }
71
72          static const int ORDER = 4;
73
74          vdouble x;
75          vdouble y;
76          vdouble z;
77  };
```

Метод Адамса

```
1       class AdamsMethod {
2           public:
3               AdamsMethod(CauchyProblem t) :
4                   f(t.f),
5                   left(t.left),
6                   right(t.right),
7                   y0(t.y0),
8                   z0(t.z0),
9                   task(t) {}
10
11              void operator() (double h) {
12                  RungeKutta2 rk(task);
```

```
            rk(h);
            x = GetValInRange(left, right, h);
            y = rk.GetY();
            z = rk.GetZ();

            y.resize(4);
            z.resize(4);

            int n = x.size();
            for (int i = 3; i < n - 1; ++i) {
                double z_next = z[i] + h / 24 * df(i);
                z.push_back(z_next);
                double y_next = y[i] + h / 24 * dg(i);
                y.push_back(y_next);
            }
        }

        vdouble GetY() {
            return y;
        }

        vdouble GetZ() {
            return z;
        }
    private:

        double df(int i) {
            double f0 = (*f)(x[i], y[i], z[i]);
            double f1 = (*f)(x[i - 1], y[i - 1], z[i - 1]);
            double f2 = (*f)(x[i - 2], y[i - 2], z[i - 2]);
            double f3 = (*f)(x[i - 3], y[i - 3], z[i - 3]);

            return (55*f0 - 59*f1 + 37*f2 - 9*f3);
        }

        double dg(int i) {
            return (55*z[i] - 59*z[i - 1] + 37*z[i - 2] - 9*z[i - 3]);
        }

        CauchyProblem task;
        Function3* f;
        double left;
        double right;
        double y0;
        double z0;

        vdouble x;
        vdouble y;
        vdouble z;
```

```
62 ||      };
```

Результаты

Пример работы программы

```
(base) MacBook-Air-Dima:Program dandachok$ ./part1 <tests/input/part1.t
Real | Eiler | RK | Adams |RK(RR) |Delta
3 | 3 | 3 | 3 |3 |0
3.02975 | 3 | 3.03008 | 3.03008
3.11997 | 3.06 | 3.12134 | 3.12134 |3.12135 |0.0013708
3.27373 | 3.1804 | 3.27689 | 3.27689
3.49633 | 3.36367 | 3.50213 | 3.50203 |3.50213 |0.00579732
3.79575 | 3.61449 | 3.80519 | 3.80478
4.18322 | 3.94008 | 4.19752 | 4.19668 |4.19753 |0.0143053
4.67419 | 4.35077 | 4.69485 | 4.69328
5.2897 | 4.86086 | 5.31861 | 5.31591 |5.31863 |0.0289321
6.05839 | 5.48985 | 6.09796 | 6.09351
7.01963 | 6.26441 | 7.07297 | 7.06584 |7.07303 |0.0533953
(base) MacBook-Air-Dima:Program dandachok$
```

Real	Eiler	RK	Adams	RK(RR)	Delta
3	3	3	3	3	0
3.02975	3	3.03008	3.03008		
3.11997	3.06	3.12134	3.12134	3.12135	0.0013708
3.27373	3.1804	3.27689	3.27689		
3.49633	3.36367	3.50213	3.50203	3.50213	0.00579732
3.79575	3.61449	3.80519	3.80478		
4.18322	3.94008	4.19752	4.19668	4.19753	0.0143053
4.67419	4.35077	4.69485	4.69328		
5.2897	4.86086	5.31861	5.31591	5.31863	0.0289321
6.05839	5.48985	6.09796	6.09351		
7.01963	6.26441	7.07297	7.06584	7.07303	0.0533953

Часть 2

Задание

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: 3

Задача Коши:

$$x^2(x+1)y'' - 2y = 0$$
$$y'(1) = -1$$
$$2y(2) - 4y'(2) = 4$$

Точное решение:

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

Теория

Метод стрельбы

Суть метода заключена в многократном решении задачи Коши для приближенного нахождения решения краевой задачи.

Пусть надо решить краевую задачу на отрезке. Вместо исходной задачи формулируется задача Коши с начальными условиями

$$y(a) = \eta$$
$$y'(b) = y_0$$

Задачу можно сформулировать таким образом: требуется найти такое значение переменной η, чтобы решение $y(a, y_0, \eta)$ в правом конце отрезка совпало со значением из начальных условий. Другими словами, решение исходной задачи эквивалентно нахождению корня уравнения

$$\Phi(\eta) = 0$$

где $\Phi(\eta) = \alpha_1 y + \alpha_2 y' - \beta$, α_1, α_2, β - коэффициенты уравнения правой границы.

Следующее значение искомого корня определяется по соотношению

$$\eta_{j+2} = \eta_{j+1} - \frac{\eta_{j+1} - \eta_j}{\Phi(\eta_{j+1}) - \Phi(\eta_j)}\Phi(\eta_{j+1})$$

Конечно-разностный метод

Рассмотрим двухточечную краевую задачу для линейного дифференциального уравнения второго порядка на отрезке $[a, b]$:

$$y'' + p(x)y' + q(x)y = f(x)$$
$$y'(a) = z_0$$
$$\alpha_1 y(b) + \alpha_2 y'(b) = \beta$$

Введем разностную аппроксимацию производных следующим образом

$$y'_k = \frac{y_{k+1} - y_{k-1}}{2h} + O(h^2)$$
$$y''_k = \frac{y_{k+1} - 2y_k + y_{k-1}}{h^2} + O(h^2)$$

Подставляя аппроксимации, приводя подобные и учитывая граничные условия, получим систему линейных алгебраических уравнений с трехдиагональной матрицей коэффициентов

$$\begin{cases} -y_1 + y_2 = hz_0 \\ \left(1 - \dfrac{hp(x_k)}{2}\right) y_{k-1} + (h^2 q(x_k) - 2) y_k + \left(1 + \dfrac{hp(x_k)}{2}\right) y_{k+1} = h^2 f(x_k), \qquad k = 2, \ldots, N-2 \\ -\alpha_1 y_{N-1} + (h\alpha_2 + \alpha_1) y_N = h\beta \end{cases}$$

Реализация

Метод стрельбы

```
class ShootingMethodL2R3 {
    public:
        ShootingMethodL2R3(BVProblemL2R3 task) : task(task), f(task.f) {}

        void Calc (double h, double eps) {
            x = GetValInRange(task.left, task.right, h);
            CauchyProblem ctask1 = task;
            CauchyProblem ctask2 = task;
            ctask1.y0 = 1;
            ctask2.y0 = 0;
            RungeKutta2 rg1(ctask1);
            RungeKutta2 rg2(ctask2);
            double& n1 = rg1.y0;
            double& n2 = rg2.y0;
            vdouble y1 = rg1(h);
            vdouble y2 = rg2(h);
            if (isFinish(y1, eps)) {
                y = y1;
                return;
            } else if (isFinish(y2, eps)) {
                y = y2;
                return;
            }
            do {
                double next_n = GetNextN(n1, n2, y1, y2);
                n1 = n2;
                n2 = next_n;
                y1 = y2;
                y2 = rg2(h);
            } while (!isFinish(y2, eps));

            y = y2;
        }

        vdouble GetY() const {
            return y;
        }

    private:

        double GetYDer(const vdouble& y, double xp) {
            int i = 0;
            for (; i < x.size() - 2; ++i) {
                if (x[i] <= xp && xp <= x[i + 1]) {
                    break;
```

```
46                }
47            }
48            return (y[i + 1] - y[i]) / (x[i + 1] - x[i]);
49        }
50
51        double GetPhi (const vdouble& y) {
52            double y_der = GetYDer(y, task.right);
53            double phi = task.re.b * y_der + task.re.a * y[y.size() - 1] - task.re.c;
54            return phi;
55        }
56
57        double GetNextN(double n1, double n2, const vdouble& y1, const vdouble& y2) {
58            double phi1 = GetPhi(y1);
59            double phi2 = GetPhi(y2);
60
61            return n2 - (n2 - n1) / (phi2 - phi1) * phi2;
62        }
63
64        bool isFinish(const vdouble& y, double eps) {
65            double phi = GetPhi(y);
66            return std::abs(phi) < eps;
67        }
68
69        Function3* f;
70        BVProblemL2R3 task;
71
72        vdouble x;
73        vdouble y;
74    };
```

Конечно-разностный метод

```
1   class FDMethod {
2       public:
3       FDMethod(Function1* p, Function1* q, BVProblemL2R3 t) :
4           p(p), q(q), task(t) {}
5
6       void Calc(double h) {
7           x = GetValInRange(task.left, task.right, h);
8           int n = (task.right - task.left) / h;
9           vvdouble mat(n + 1, vdouble(4));
10          mat[0][0] = 0;
11          mat[0][1] = -1;
12          mat[0][2] = 1;
13          mat[0][3] = task.z0 * h;
14
15          for (int i = 1; i < n; ++i) {
16              mat[i][0] = 1 - (*p)(x[i]) * h / 2;
17              mat[i][1] = (*q)(x[i]) * h * h - 2;
```

```
18            mat[i][2] = 1 + (*p)(x[i]) * h / 2;
19            mat[i][3] = 0;
20        }
21
22        mat[n][0] = -task.re.b;
23        mat[n][1] = h * task.re.a + task.re.b;
24        mat[n][2] = 0;
25        mat[n][3] = task.re.c * h;
26
27        y = SweepMethod(mat);
28    }
29
30    vdouble GetY() {
31        return y;
32    }
33
34    private:
35        Function1* p;
36        Function1* q;
37        BVProblemL2R3 task;
38
39        vdouble x;
40        vdouble y;
41 };
```

Результаты

```
(base) MacBook-Air-Dima:Program dandachok$
2 1.93427 0.065731 2.08858 0.0885836
1.90909 1.84306 0.066033 1.98858 0.0794927
1.83333 1.76647 0.0668585 1.90109 0.0677539
1.76923 1.70112 0.0681084 1.82307 0.0538367
1.71429 1.64458 0.0697104 1.75237 0.0380841
1.66667 1.59506 0.0716098 1.68742 0.0207553
1.625 1.55124 0.0737647 1.62705 0.00205174
1.58824 1.51209 0.0761418 1.57037 0.0178668
1.55556 1.47684 0.0787144 1.51669 0.0388704
1.52632 1.44485 0.0814609 1.46546 0.0608523
1.5 1.41564 0.0843633 1.41628 0.0837226
(base) MacBook-Air-Dima:Program dandachok$
```

Точное	Метод стрельбы	Погрешность	Конечно-разностный метод	Погрешность
2	1.93427	0.065731	2.08858	0.0885836
1.90909	1.84306	0.066033	1.98858	0.0794927
1.83333	1.76647	0.0668585	1.90109	0.0677539
1.76923	1.70112	0.0681084	1.82307	0.0538367
1.71429	1.64458	0.0697104	1.75237	0.0380841
1.66667	1.59506	0.0716098	1.68742	0.0207553
1.625	1.55124	0.0737647	1.62705	0.00205174
1.58824	1.51209	0.0761418	1.57037	0.0178668
1.55556	1.47684	0.0787144	1.51669	0.0388704
1.52632	1.44485	0.0814609	1.46546	0.0608523
1.5	1.41564	0.0843633	1.41628	0.0837226