

**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: С. П. Сабурова
Преподаватель: Д. Е. Пивоваров
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: 18

18	$y'' - \frac{x+1}{x} y' - 2 \frac{x-1}{x} y = 0,$ $y(1) = 1,$ $y'(1) = 1,$ $x \in [1, 2], h = 0.1$	$y = \frac{e^{2x}}{3e^2} + \frac{(3x+1)e^{-x}}{3e}$
----	--	---

Рис. 1: Входные данные

2 Результаты работы

0	1	1	0	0
0.1	0.999997214	1.02015909	0.0201618745	4.85127383e-10
0.2	0.999954553	1.08258217	0.0826276116	1.99890654e-09
0.3	0.999762901	1.19339076	0.193627855	5.2976775e-09
0.4	0.999217118	1.36378949	0.364572372	1.24106849e-08
0.5	0.997969961	1.61177538	0.613805422	2.92756729e-08
0.6	0.99543369	1.96499492	0.96956123	7.75003038e-08
0.7	0.990543029	2.46534923	1.4748062	2.72803673e-07
0.8	0.980994803	3.17635873	2.19536392	2.00109877e-06
0.9	0.959704277	4.19498468	3.2352804	-0.000340870108
1	0.94941541	5.67077427	4.72135886	-0.00267666498

Рис. 2: Вывод программы в консоли

0		1		1		0		0
0.1		0.999997214		1.02015909		0.0201618745		4.85127383e-10
0.2		0.999954553		1.08258217		0.0826276116		-6.5059492e-09
0.3		0.999762901		1.19339076		0.193627855		-6.18247574e-08
0.4		0.999227277		1.36378949		0.364562213		4.91268934e-07
0.5		0.998019586		1.61177538		0.613755797		2.85296911e-06
0.6		0.995562559		1.96499492		0.969432361		7.50172358e-06
0.7		0.990882335		2.46534923		1.4744669		1.96942549e-05
0.8		0.98204837		3.17635873		2.19431036		5.90069164e-05
0.9		0.964285616		4.19498468		3.23069906		0.000305874024
1		0.914952815		5.67077427		4.75582146		-0.000311150979

Рис. 3: Вывод программы в консоли

4.2 Метод стрельбы и конечно-разностный метод

3 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: 18

18	$x y'' - (x+1)y' - 2(x-1)y = 0,$ $y'(0) = 4,$ $y'(1) - 2y(1) = -9e^{-1}$	$y(x) = e^{2x} + (3x+1)e^{-x}$
----	--	--------------------------------

Рис. 4: Входные данные

4 Результаты работы

```
Method shooting:
  x | y | 6.00033546 | 12.0003355 | 0
-----
2.1 | -5.62947965 | 6.30014775 | 11.9296274 | 7.11083045e-07
2.2 | -5.25387502 | 6.60006252 | 11.8539375 | 1.52207649e-06
2.3 | -4.86523307 | 6.90002542 | 11.7652585 | 2.4384716e-06
2.4 | -4.45271541 | 7.20000993 | 11.6527253 | 3.45004839e-06
2.5 | -4.00130004 | 7.50000373 | 11.5013038 | 4.5139474e-06
2.6 | -3.48985959 | 7.80000134 | 11.2898609 | 5.52371095e-06
2.7 | -2.88829474 | 8.10000047 | 10.9882952 | 6.25339757e-06
2.8 | -2.15322423 | 8.40000015 | 10.5532244 | 6.25750622e-06
2.9 | -1.22145374 | 8.70000005 | 9.92145379 | 4.69259332e-06
3 | -3.10862447e-15 | 9.00000002 | 9.00000002 | -6.66133815e-17
```

Рис. 5: Вывод программы в консоли

```
Finite difference method:
  x | y | 6.00033546 | 12.0003355 | 0
-----
2.1 | -5.4 | 6.30014775 | 11.7001477 | -1.18423789e-15
2.2 | -4.8 | 6.60006252 | 11.4000625 | -2.07241631e-15
2.3 | -4.2 | 6.90002542 | 11.1000254 | -3.25665421e-15
2.4 | -3.6 | 7.20000993 | 10.8000099 | -2.812565e-15
2.5 | -3 | 7.50000373 | 10.5000037 | -2.07241631e-15
2.6 | -2.4 | 7.80000134 | 10.2000013 | -1.33226763e-15
2.7 | -1.8 | 8.10000047 | 9.90000047 | -5.18104078e-16
2.8 | -1.2 | 8.40000015 | 9.60000015 | -2.96059473e-16
2.9 | -0.6 | 8.70000005 | 9.30000005 | -1.48029737e-16
3 | 0 | 9.00000002 | 9.00000002 | 0
```

Рис. 6: Вывод программы в консоли

5 Исходный код

```
1  #if !defined(MATRIX)
2  #define MATRIX
3
4  #include <ccomplex>
5  #include <cmath>
6  #include <fstream>
7  #include <iostream>
8  #include <vector>
9
10 using namespace std;
11
12 #define EPS 1e-5
13
14 class Matrix {
15 private:
16     int rows_, cols_;
17     vector<vector<double>> matrix_;
18     vector<int> swp_;
19
20     void SwapMatrix(Matrix &other) {
21         swap(rows_, other.rows_);
22         swap(cols_, other.cols_);
23         swap(matrix_, other.matrix_);
24     }
25     Matrix Minor(int i, int j) const {
26         Matrix result(rows_ - 1, cols_ - 1);
27         int ki = 0;
28         for (int new_i = 0; new_i < result.rows_; ++new_i) {
29             int kj = 0;
30             if (new_i == i)
31                 ki = 1;
32             for (int new_j = 0; new_j < result.cols_; ++new_j) {
33                 if (new_j == j)
34                     kj = 1;
35                 if (new_i + ki < rows_ && new_j + kj < cols_)
36                     result.matrix_[new_i][new_j] = matrix_[new_i + ki][new_j + kj];
37             }
38         }
39         return result;
40     }
41
42 public:
43     Matrix(int rows, int cols) {
44         if (rows < 1 || cols < 1)
45             throw runtime_error(
46                 "          \n");
47         rows_ = rows;
```



```

48     cols_ = cols;
49     matrix_.resize(rows_);
50     for (int i = 0; i < rows_; ++i) {
51         matrix_[i].resize(cols_);
52     }
53 }
54 Matrix() : Matrix(1, 1) {}
55 Matrix(const Matrix &other) : Matrix(other.rows_, other.cols_) {
56     for (int i = 0; i < rows_; ++i) {
57         for (int j = 0; j < cols_; ++j) {
58             matrix_[i][j] = other.matrix_[i][j];
59         }
60     }
61 }
62 Matrix(Matrix &&other) {
63     this->SwapMatrix(other);
64     other.rows_ = 0;
65     other.cols_ = 0;
66 }
67
68 int GetRows() const { return rows_; }
69 int GetCols() const { return cols_; }
70 const vector<int> &GetSwp() const { return swp_; }
71
72 void SetRows(int rows) {
73     if (rows < 1)
74         throw runtime_error("    \n");
75     Matrix tmp_matrix(rows, cols_);
76     for (int i = 0; i < min(tmp_matrix.rows_, rows_); ++i) {
77         for (int j = 0; j < cols_; ++j) {
78             tmp_matrix.matrix_[i][j] = matrix_[i][j];
79         }
80     }
81     this->SwapMatrix(tmp_matrix);
82 }
83 void SetCols(int cols) {
84     if (cols < 1)
85         throw runtime_error("    \n");
86     Matrix tmp_matrix(rows_, cols);
87     for (int i = 0; i < rows_; ++i) {
88         for (int j = 0; j < min(tmp_matrix.cols_, cols_); ++j) {
89             tmp_matrix.matrix_[i][j] = matrix_[i][j];
90         }
91     }
92     this->SwapMatrix(tmp_matrix);
93 }
94
95 bool EqMatrix(const Matrix &other) const {
96     bool flag = true;

```

```

97     if (rows_ != other.rows_ || cols_ != other.cols_)
98         flag = false;
99     else {
100         for (int i = 0; i < rows_; ++i) {
101             for (int j = 0; j < cols_; ++j) {
102                 if (fabs(matrix_[i][j] - other.matrix_[i][j]) > EPS)
103                     flag = false;
104             }
105         }
106     }
107     return flag;
108 }
109 void SumMatrix(const Matrix &other) {
110     if (rows_ != other.rows_ || cols_ != other.cols_)
111         throw runtime_error("    \n");
112     for (int i = 0; i < rows_; ++i) {
113         for (int j = 0; j < cols_; ++j) {
114             matrix_[i][j] += other.matrix_[i][j];
115         }
116     }
117 }
118 void SubMatrix(const Matrix &other) {
119     if (rows_ != other.rows_ || cols_ != other.cols_)
120         throw runtime_error("    \n");
121     for (int i = 0; i < rows_; ++i) {
122         for (int j = 0; j < cols_; ++j) {
123             matrix_[i][j] -= other.matrix_[i][j];
124         }
125     }
126 }
127 void MulNumber(const double num) {
128     for (int i = 0; i < rows_; ++i) {
129         for (int j = 0; j < cols_; ++j) {
130             matrix_[i][j] *= num;
131         }
132     }
133 }
134
135 Matrix MulMatrixReturn(const double num) {
136     Matrix tmp = *this;
137     tmp.MulNumber(num);
138     return tmp;
139 }
140
141 void MulMatrix(const Matrix &other) {
142     if (cols_ != other.rows_)
143         throw runtime_error("    \n");
144     Matrix tmp(rows_, other.cols_);
145     for (int i = 0; i < rows_; ++i) {

```

```

146         for (int j = 0; j < other.cols_; ++j) {
147             for (int k = 0; k < cols_; ++k)
148                 tmp.matrix_[i][j] += matrix_[i][k] * other.matrix_[k][j];
149         }
150     }
151     this->SwapMatrix(tmp);
152 }
153
154 Matrix MulMatrixReturn(const Matrix &other) {
155     Matrix tmp = *this;
156     tmp.MulMatrix(other);
157     return tmp;
158 }
159
160 Matrix Transpose() const {
161     Matrix result(cols_, rows_);
162     for (int i = 0; i < result.rows_; ++i) {
163         for (int j = 0; j < result.cols_; ++j) {
164             result.matrix_[i][j] = matrix_[j][i];
165         }
166     }
167     return result;
168 }
169
170 pair<Matrix, Matrix> LU() {
171     swp_.clear();
172     int n = this->GetRows();
173     Matrix U(*this);
174     Matrix L(n, n);
175     for (int k = 0; k < n; ++k) { // k - , ,
176         int index = k; // index - max k
177         for (int i = k + 1; i < n; ++i) {
178             if (abs(U(i, k)) > abs(U(index, k))) {
179                 index = i;
180             }
181         }
182         swap(U(k), U(index));
183         swap(L(k), L(index));
184         swp_.push_back(index);
185         for (int i = k + 1; i < n; ++i) {
186             double m = U(i, k) / U(k, k);
187             L(i, k) = m;
188             for (int j = k; j < n; ++j) {
189                 U(i, j) -= m * U(k, j);
190             }
191         }
192     }
193     for (int i = 0; i < n; ++i) {
194         L(i, i) = 1;

```



```

195     }
196     return {L, U};
197 }
198
199 Matrix Solve(Matrix &C, Matrix &L, Matrix &U) {
200     Matrix B(C);
201     vector<int> swp = this->GetSwp();
202     for (int i = 0; i < swp.size(); ++i) {
203         swap(B(i), B(swp[i]));
204     }
205     int n = this->GetRows();
206     // LUx = b
207     // Lz = b
208     Matrix Z(n, 1);
209     for (int i = 0; i < n; ++i) {
210         Z(i, 0) = B(i, 0);
211         for (int j = 0; j < i; ++j) {
212             Z(i, 0) -= L(i, j) * Z(j, 0);
213         }
214     }
215     // Ux = z
216     Matrix X(n, 1);
217     for (int i = n - 1; i >= 0; --i) {
218         X(i, 0) = Z(i, 0);
219         for (int j = i + 1; j < n; ++j) {
220             X(i, 0) -= U(i, j) * X(j, 0);
221         }
222         X(i, 0) = X(i, 0) / U(i, i);
223     }
224     return X;
225 }
226
227 Matrix Solve(Matrix &C) {
228     auto [L, U] = this->LU();
229     return this->Solve(C, L, U);
230 }
231
232 double Determinant() {
233     // detA = det(LU) = detL * detU = detU
234     double result = 1;
235     auto [L, U] = this->LU();
236     for (int i = 0; i < rows_; ++i) {
237         result *= U(i, i);
238     }
239     // swap
240     int sign = 0;
241     vector<int> swp = this->GetSwp();
242     for (int i = 0; i < swp.size(); ++i) {
243         if (swp[i] != i)

```

```

244         ++sign;
245     }
246     if (sign % 2 != 0)
247         result = -result;
248     return result;
249 }
250
251 Matrix InverseMatrix() {
252     int n = this->GetRows();
253     Matrix B(n, 1);
254     Matrix result(n, n);
255     for (int i = 0; i < n; ++i) {
256         if (i > 0)
257             B(i - 1, 0) = 0;
258         B(i, 0) = 1;
259         auto [L, U] = this->LU();
260         Matrix res_i = this->Solve(B, L, U);
261         for (int k = 0; k < n; ++k)
262             result(k, i) = res_i(k, 0);
263     }
264     return result;
265 }
266
267 Matrix CalcComplements() const {
268     Matrix result(*this);
269     if (rows_ != cols_)
270         throw runtime_error(
271             "
272             "\n");
273     if (rows_ == 1) {
274         result.matrix_[0][0] = 1;
275     } else {
276         for (int i = 0; i < rows_; ++i) {
277             for (int j = 0; j < cols_; ++j) {
278                 Matrix new_matrix = this->Minor(i, j);
279                 double minor_det = new_matrix.Determinant();
280                 result.matrix_[i][j] = pow(-1, i + j) * minor_det;
281             }
282         }
283     }
284     return result;
285 }
286
287 Matrix run_through_method(Matrix &B) {
288     int n = this->rows_;
289     vector<double> P, Q; //  $x_n = P_n * x_{n+1} + Q_n$ 
290     P.push_back((-1) * (*this)(0, 1) / (*this)(0, 0)); //  $P[0] = -c1/b1$ 
291     Q.push_back(B(0, 0) / (*this)(0, 0)); //  $Q[0] = d1/b1$ 
292     for (int i = 1; i < n; ++i) {

```

```

293     if (i == n - 1) {
294         P.push_back(0); // c_n = 0
295     } else {
296         P.push_back((-1) * (*this)(i, i + 1) / ((*this)(i, i) + (*this)(i, i -
297             1) * P[i - 1])); // P_i = -c_i / (b_i + a_i * P_{i-1})
298     }
299     Q.push_back((B(i, 0) - (*this)(i, i - 1) * Q[i - 1]) / ((*this)(i, i) + (*
300         this)(i, i - 1) * P[i - 1])); // Q_i = (d_i - a_i * Q_{i-1}) / (b_i + a_i
301         * P_{i-1})
302 }
303 Matrix X(n, 1);
304 X(n - 1, 0) = Q[n - 1];
305 for (int i = n - 2; i >= 0; --i) {
306     X(i, 0) = P[i] * X(i + 1, 0) + Q[i];
307 }
308 return X;
309 }
310
311 double norm() {
312     double res = 0;
313     for (int i = 0; i < this->rows_; ++i) {
314         double tmp_res = 0;
315         for (int j = 0; j < this->cols_; ++j) {
316             tmp_res += abs((*this)(i, j));
317         }
318         if (tmp_res > res)
319             res = tmp_res;
320     }
321     return res;
322 }
323
324 pair<Matrix, int> simple_iterations(Matrix &B, double eps) {
325     int n = this->rows_;
326     Matrix Alpha(n, n);
327     Matrix Beta(n, 1);
328     for (int i = 0; i < n; ++i) {
329         for (int j = 0; j < n; ++j) {
330             Alpha(i, j) = (-1) * (*this)(i, j) / (*this)(i, i);
331         }
332         Alpha(i, i) = 0;
333         Beta(i, 0) = B(i, 0) / (*this)(i, i);
334     }
335     Matrix X(n, 1), Prev_X(n, 1);
336     int k = 1;
337     Prev_X = Beta;
338     X = Beta + Alpha * Prev_X;
339     // eps_k = ||Alpha|| / (1 - ||Alpha||) * ||x_k - x_{k-1}||
340     double eps_k = 0;
341     double norm = Alpha.norm();

```

```

339     if (norm >= 1) {
340         eps_k = (X - Prev_X).norm();
341     } else {
342         eps_k = norm / (1 - norm) * (X - Prev_X).norm();
343     }
344     while (eps_k > eps) { // eps_k <= eps
345         Prev_X = X;
346         X = Beta + Alpha * X;
347         if (norm >= 1) {
348             eps_k = (X - Prev_X).norm();
349         } else {
350             eps_k = norm / (1 - norm) * (X - Prev_X).norm();
351         }
352         ++k;
353     }
354     return {X, k};
355 }
356
357 pair<Matrix, int> seidel(Matrix &R, double eps) {
358     int n = this->rows_;
359     Matrix Alpha(n, n), Beta(n, 1), E(n, n);
360     for (int i = 0; i < n; ++i) {
361         for (int j = 0; j < n; ++j) {
362             Alpha(i, j) = (-1) * (*this)(i, j) / (*this)(i, i);
363         }
364         Alpha(i, i) = 0;
365         E(i, i) = 1;
366         Beta(i, 0) = R(i, 0) / (*this)(i, i);
367     }
368     // Alpha = B + C
369     Matrix C(n, n), B(n, n);
370     for (int i = 0; i < n; ++i) {
371         for (int j = 0; j < n; ++j) {
372             if (j < i)
373                 B(i, j) = Alpha(i, j);
374             else
375                 C(i, j) = Alpha(i, j);
376         }
377     }
378     //  $x_{k+1} = (E - B)^{-1} * C * x_k + (E - B)^{-1} * Beta$ 
379     Matrix X(n, 1), Prev_X(n, 1);
380     int k = 1;
381     Prev_X = Beta;
382     Matrix Tmp_Beta = (E - B).InverseMatrix() * Beta;
383     Matrix Tmp_Alpha = (E - B).InverseMatrix() * C;
384     X = Tmp_Alpha * Prev_X + Tmp_Beta;
385     double eps_k = 0;
386     double norm = Alpha.norm();
387     if (norm >= 1) {

```

```

388         eps_k = (X - Prev_X).norm();
389     } else {
390         eps_k = C.norm() / (1 - norm) * (X - Prev_X).norm();
391     }
392     // eps_k = ||C|| / (1 - ||Alpha||) * ||x_k - x_{k-1}||
393     while (eps_k > eps) {
394         Prev_X = X;
395         X = Tmp_Alpha * Prev_X + Tmp_Beta;
396         if (norm >= 1) {
397             eps_k = (X - Prev_X).norm();
398         } else {
399             eps_k = C.norm() / (1 - norm) * (X - Prev_X).norm();
400         }
401         ++k;
402     }
403     return {X, k};
404 }
405
406 // helper
407 double sum_square() {
408     int n = this->rows_;
409     double res = 0;
410     //
411     for (int i = 0; i < n; ++i) {
412         for (int j = 0; j < n; ++j) {
413             if (i != j)
414                 res += (*this)(i, j) * (*this)(i, j);
415         }
416     }
417     return sqrt(res);
418 }
419
420 pair<pair<Matrix, Matrix>, int> jacobi_method(double eps) {
421     int n = this->rows_;
422     int k = 0;
423     pair<int, int> max_index;
424     Matrix A = *this;
425     Matrix Self_Vectors(n, n);
426     for (int i = 0; i < n; ++i) {
427         Self_Vectors(i, i) = 1;
428     }
429     while (A.sum_square() > eps) {
430         Matrix U(n, n);
431         max_index = {1, 0}; // index abs(max_elem)
432         for (int i = 0; i < n; ++i) {
433             for (int j = 0; j < n; ++j) {
434                 if (i != j && abs(A(i, j)) > abs(A(max_index.first, max_index.second)))
435                     max_index = {i, j};

```

```

436     }
437 }
438 for (int i = 0; i < n; ++i) {
439     U(i, i) = 1;
440 }
441 // phi = 1/2 * arctg (2 * a(i, j) / (a(i, i) - a(j, j)))
442 // phi = PI/4, a(i, i) = a(j, j)
443 double phi;
444 if (A(max_index.first, max_index.first) == A(max_index.second, max_index.
    second))
445     phi = M_PI / 4;
446 else
447     phi = 0.5 * atan(2 * A(max_index.first, max_index.second) / (A(max_index
    .first, max_index.first) - A(max_index.second, max_index.second)));
448 U(max_index.first, max_index.first) = cos(phi);
449 U(max_index.first, max_index.second) = (-1) * sin(phi);
450 U(max_index.second, max_index.first) = sin(phi);
451 U(max_index.second, max_index.second) = cos(phi);
452 Matrix U_T = U.Transpose();
453 // A^{k+1} = U_T^k * A^k * U^k
454 A = U_T.MulMatrixReturn(A).MulMatrixReturn(U);
455 Self_Vectors.MulMatrix(U);
456 ++k;
457 }
458 return {{A, Self_Vectors}, k};
459 }
460
461 int sign(double x) {
462     if (x > 0)
463         return 1;
464     if (x < 0)
465         return -1;
466     return 0;
467 }
468
469 pair<Matrix, Matrix> qr_decomposition() {
470     int n = this->rows_;
471     Matrix E(n, n);
472     for (int i = 0; i < n; ++i) {
473         E(i, i) = 1;
474     }
475     Matrix Q = E;
476     Matrix A = *this;
477     for (int i = 0; i < n - 1; ++i) {
478         Matrix H(n, n);
479         Matrix V(n, 1);
480         // v_1 = a_11 + sign(a11) * || 1 ||
481         // v_i = a_i1
482         double norm = 0;

```



```

483     for (int j = i; j < n; ++j) {
484         norm += A(j, i) * A(j, i);
485     }
486     norm = sqrt(norm);
487     for (int j = i; j < V.GetRows(); ++j) {
488         if (j == i) {
489             V(j, 0) = A(i, i) + sign(A(i, i)) * norm;
490         } else {
491             V(j, 0) = A(j, i);
492         }
493     }
494     Matrix V_T = V.Transpose();
495     //  $H = E - 2 * v * v_t / (v_t * v)$ 
496     H = E - V.MulMatrixReturn(V_T).MulMatrixReturn(2 / (V_T.MulMatrixReturn(V))
497         (0, 0));
498     A = H.MulMatrixReturn(A);
499     Q = Q.MulMatrixReturn(H);
500 }
501 //  $Q^{-1} = Q_T$ 
502 return {Q, A};
503 }
504
505 vector<complex<double>> qr_method(double eps) {
506     int n = this->rows_;
507     Matrix A = *this;
508     vector<complex<double>> lambda;
509     vector<complex<double>> lambda_prev;
510     int counter = 0;
511     int iter = 50;
512     while (true) {
513         auto [Q, R] = A.qr_decomposition();
514         A = R.MulMatrixReturn(Q);
515         // cout << "A\n";
516         // A.ShowMatrix();
517         if (counter != iter) {
518             ++counter;
519             continue;
520         }
521         for (int i = 0; i < n; i += 1) {
522             double sum = 0;
523             for (int j = i + 1; j < n; ++j) {
524                 sum += abs(A(j, i));
525             }
526             if (sum < 0.001) {
527                 lambda.push_back(A(i, i));
528             } else {
529                 //  $(a_{jj} - \text{Lambda})(a_{j+1,j+1} - \text{Lambda}) = a_{j,j+1} * a_{j+1,j}$ 
530                 double a = 1;
531                 double b = (-1) * (A(i, i) + A(i + 1, i + 1));

```

```

531         double c = A(i, i) * A(i + 1, i + 1) - A(i, i + 1) * A(i + 1, i);
532         double d = b * b - 4 * c;
533         complex<double> x1, x2;
534         if (d < 0) {
535             x1 = (-b + sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
536             x2 = (-b - sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
537         } else {
538             x1 = (-b + sqrt(d)) / (2 * a);
539             x2 = (-b - sqrt(d)) / (2 * a);
540         }
541         lambda.push_back(x1);
542         lambda.push_back(x2);
543         ++i;
544     }
545 }
546 bool exit = true;
547 //
548 if (lambda_prev.size() != 0) {
549     for (int i = 0; i < lambda.size(); i++) {
550         if (abs(lambda[i] - lambda_prev[i]) > eps) {
551             exit = false;
552             break;
553         }
554     }
555     if (exit == true)
556         break;
557 }
558 lambda_prev = lambda;
559 lambda.clear();
560 counter = 0;
561 }
562 return lambda;
563 }
564
565 Matrix operator+(const Matrix &other) {
566     Matrix result(*this);
567     result.SumMatrix(other);
568     return result;
569 }
570 Matrix operator-(const Matrix &other) {
571     Matrix result(*this);
572     result.SubMatrix(other);
573     return result;
574 }
575 Matrix operator*(const Matrix &other) {
576     Matrix result(*this);
577     result.MulMatrix(other);
578     return result;
579 }

```

```

580     Matrix operator*(const double num) {
581         Matrix result(*this);
582         result.MulNumber(num);
583         return result;
584     }
585     bool operator==(const Matrix &other) { return this->EqMatrix(other); }
586     Matrix operator=(const Matrix &other) {
587         if (this != &other) { // b = b
588             Matrix tmp(other);
589             this->SwapMatrix(tmp);
590         }
591         return *this;
592     }
593     void operator+=(const Matrix &other) { this->SumMatrix(other); }
594     void operator-=(const Matrix &other) { this->SubMatrix(other); }
595     void operator*=(const Matrix &other) { this->MulMatrix(other); }
596     void operator*=(const double num) { this->MulNumber(num); }
597     double &operator()(int i, int j) {
598         if (i < 0 || i >= rows_ || j < 0 || j >= cols_)
599             throw runtime_error("    \n");
600         return matrix_[i][j];
601     }
602     vector<double> &operator()(int row) { return matrix_[row]; }
603
604     void ShowMatrix() const {
605         for (int i = 0; i < rows_; ++i) {
606             for (int j = 0; j < cols_; ++j) {
607                 cout << matrix_[i][j] << " ";
608             }
609             cout << "\n";
610         }
611     }
612 };
613
614 ostream &operator<<(ostream &stream, Matrix A) {
615     for (int i = 0; i < A.GetRows(); i++) {
616         for (int j = 0; j < A.GetCols(); j++)
617             stream << A(i, j) << ' ';
618         stream << '\n';
619     }
620     return stream;
621 }
622
623 istream &operator>>(istream &stream, Matrix &A) {
624     for (int i = 0; i < A.GetRows(); i++) {
625         for (int j = 0; j < A.GetCols(); j++)
626             stream >> A(i, j);
627     }
628     return stream;

```

```

629 }
630
631 #endif // MATRIX

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 double method_runge_romberg(double y1, double y2, int64_t p) {
6     return (y1 - y2) / (pow(2, p) - 1);
7 }
8
9 vector<double> num_vector(vector<double> a, double n) {
10     for (int i = 0; i < a.size(); ++i) {
11         a[i] *= n;
12     }
13     return a;
14 }
15
16 vector<vector<double>> method_runge_kutta_4(vector<double> (*f)(double, double, double
    ), double x_start, double x_finish, double h, double y0, double z0, int iter = -1)
    {
17     int n;
18     if (iter == -1) {
19         n = (x_finish - x_start) / h;
20     } else
21         n = iter;
22     vector<double> X(n + 1), Y(n + 1), Z(n + 1);
23     for (int i = 0; i <= n; ++i) {
24         X[i] = x_start + i * h;
25     }
26     Y[0] = y0;
27     Z[0] = z0;
28     for (int i = 1; i <= n; ++i) {
29         vector<double> K_1 = num_vector(f(X[i - 1], Y[i - 1], Z[i - 1]), h);
30         vector<double> K_2 = num_vector(f(X[i - 1] + h / 2, Y[i - 1] + K_1[0] / 2, Z[i
            - 1] + K_1[1] / 2), h);
31         vector<double> K_3 = num_vector(f(X[i - 1] + h / 2, Y[i - 1] + K_2[0] / 2, Z[i
            - 1] + K_2[1] / 2), h);
32         vector<double> K_4 = num_vector(f(X[i - 1] + h, Y[i - 1] + K_3[0], Z[i - 1] +
            K_3[1]), h);
33         Y[i] = Y[i - 1] + 1.0 / 6 * (K_1[0] + 2 * K_2[0] + 2 * K_3[0] + K_4[0]);
34         Z[i] = Z[i - 1] + 1.0 / 6 * (K_1[1] + 2 * K_2[1] + 2 * K_3[1] + K_4[1]);
35     }
36     return {X, Y, Z};
37 }
38
39 void print(double (*calc_exact_y)(double), string method, vector<double> &X_h1, vector
    <double> &Y_h1, vector<double> &Y_h2, int64_t p) {
40     cout << method << "\n"

```

```

41         << " x |" << " y |" << " y |" << "\t\t | -\n
         -----\n";
42     for (int i = 0; i < X_h1.size(); ++i) {
43         double exact_y = calc_exact_y(X_h1[i]);
44         cout << setprecision(9) << " " << X_h1[i] << " | " << Y_h1[i] << " | " <<
            exact_y << " | " << abs(exact_y - Y_h1[i]) << " | " << method_runge_romberg
            (Y_h1[i], Y_h2[2 * i], p) << endl;
45     }
46 }

1  #include "./lab4_1.h"
2
3  vector<double> f(double x, double y, double z) {
4      vector<double> F(2);
5      F[0] = z;
6      F[1] = (exp(x*x) + 4*x*z - y)/(4*x*x-3);
7      return F;
8  }
9
10 double calc_exact_y(double x) {
11     return (exp(x) + exp(-x) - 1) * exp(x*x);
12 }
13
14 pair<vector<double>, vector<double>> method_adams(double x_start, double x_finish,
    double h, double y0, double z0) {
15     int n = (x_finish - x_start) / h;
16     vector<double> X(n + 1), Y(n + 1), Z(n + 1);
17     for (int i = 0; i <= n; ++i) {
18         X[i] = x_start + i * h;
19     }
20     vector<vector<double>> res_runge_kutta = method_runge_kutta_4(f, x_start, x_finish,
        h, y0, z0, 4);
21     Y[0] = res_runge_kutta[1][0];
22     Y[1] = res_runge_kutta[1][1];
23     Y[2] = res_runge_kutta[1][2];
24     Y[3] = res_runge_kutta[1][3];
25     Z[0] = res_runge_kutta[2][0];
26     Z[1] = res_runge_kutta[2][1];
27     Z[2] = res_runge_kutta[2][2];
28     Z[3] = res_runge_kutta[2][3];
29     for (int i = 4; i <= n; ++i) {
30         vector<double> F_k = f(X[i - 1], Y[i - 1], Z[i - 1]);
31         vector<double> F_k_1 = f(X[i - 2], Y[i - 2], Z[i - 2]);
32         vector<double> F_k_2 = f(X[i - 3], Y[i - 3], Z[i - 3]);
33         vector<double> F_k_3 = f(X[i - 4], Y[i - 4], Z[i - 4]);
34         Y[i] = Y[i - 1] + h / 24 * (55 * F_k[0] - 59 * F_k_1[0] + 37 * F_k_2[0] - 9 *
            F_k_3[0]);
35         Z[i] = Z[i - 1] + h / 24 * (55 * F_k[1] - 59 * F_k_1[1] + 37 * F_k_2[1] - 9 *
            F_k_3[1]);
36     }

```

```

37     return {X, Y};
38 }
39
40 int main() {
41     double start_pos = 0, end_pos = 1, h = 0.1;
42     vector<vector<double>> res_h1 = method_runge_kutta_4(f, start_pos, end_pos, h, 1,
43         0);
44     vector<double> X = res_h1[0];
45     vector<double> Y_h1 = res_h1[1];
46     vector<vector<double>> res_h2 = method_runge_kutta_4(f, start_pos, end_pos, h/2, 1,
47         0);
48     vector<double> Y_h2 = res_h2[1];
49     print(calc_exact_y, " -:", X, Y_h1, Y_h2, 4);
50     auto [X2_h1, Y2_h1] = method_adams(start_pos, end_pos, h, 1, 0);
51     auto [X2_h2, Y2_h2] = method_adams(start_pos, end_pos, h/2, 1, 0);
52     print(calc_exact_y, "\n :", X2_h1, Y2_h1, Y2_h2, 4);
53     return 0;
54 }
55
56 #include "./lab4_1.h"
57
58 vector<double> f(double x, double y, double z) {
59     vector<double> F(2);
60     F[0] = z;
61     F[1] = (exp(x*x) + 4*x*z - y)/(4*x*x-3);
62     return F;
63 }
64
65 double calc_exact_y(double x) {
66     return (exp(x) + exp(-x) - 1) * exp(x*x);
67 }
68
69 pair<vector<double>, vector<double>> method_euler(double x_start, double x_finish,
70     double h, double y0, double z0) {
71     int n = (x_finish - x_start) / h;
72     vector<double> X(n + 1), Y(n + 1), Z(n + 1);
73     for (int i = 0; i <= n; ++i) {
74         X[i] = x_start + i * h;
75     }
76     Y[0] = y0;
77     Z[0] = z0;
78     for (int i = 1; i <= n; ++i) {
79         vector<double> F = f(X[i - 1], Y[i - 1], Z[i - 1]);
80         Y[i] = Y[i - 1] + h * F[0];
81         Z[i] = Z[i - 1] + h * F[1];
82     }
83     return {X, Y};
84 }
85
86 pair<vector<double>, vector<double>> first_improved_method_euler(double x_start,

```



```

31     double x_finish, double h, double y0, double z0) {
32     h = h / 2;
33     int n = (x_finish - x_start) / h;
34     vector<double> X(n + 1), Y(n + 1), Z(n + 1);
35     for (int i = 0; i <= n; ++i) {
36         X[i] = x_start + i * h;
37     }
38     Y[0] = y0;
39     Z[0] = z0;
40     for (int i = 1; i <= n; ++i) {
41         vector<double> F = f(X[i - 1], Y[i - 1], Z[i - 1]);
42         if (i % 2 == 0) {
43             Y[i] = Y[i - 2] + h * 2 * F[0];
44             Z[i] = Z[i - 2] + h * 2 * F[1];
45         } else {
46             Y[i] = Y[i - 1] + h * F[0];
47             Z[i] = Z[i - 1] + h * F[1];
48         }
49     }
50     return {X, Y};
51 }
52 pair<vector<double>, vector<double>> method_euler_recalculation(double x_start, double
53     x_finish, double h, double y0, double z0) {
54     int n = (x_finish - x_start) / h;
55     vector<double> X(n + 1), Y(n + 1), Y_tmp(n + 1), Z(n + 1), Z_tmp(n + 1);
56     for (int i = 0; i <= n; ++i) {
57         X[i] = x_start + i * h;
58     }
59     Y[0] = y0;
60     Z[0] = z0;
61     for (int i = 1; i <= n; ++i) {
62         vector<double> F_tmp = f(X[i - 1], Y[i - 1], Z[i - 1]);
63         Y_tmp[i] = Y[i - 1] + h * F_tmp[0];
64         Z_tmp[i] = Z[i - 1] + h * F_tmp[1];
65         vector<double> F = f(X[i], Y_tmp[i], Z_tmp[i]);
66         Y[i] = Y[i - 1] + h * (F_tmp[0] + F[0]) / 2;
67         Z[i] = Z[i - 1] + h * (F_tmp[1] + F[1]) / 2;
68     }
69     return {X, Y};
70 }
71 int main() {
72     auto [X_h1, Y_h1] = method_euler(1, 2, 0.1, 1, 1);
73     auto [X_h2, Y_h2] = method_euler(1, 2, 0.05, 1, 1);
74     print(calc_exact_y, "Method Euler:", X_h1, Y_h1, Y_h2, 1);
75     auto [X2_h1, Y2_h1] = first_improved_method_euler(1, 2, 0.1, 1, 1);
76     auto [X2_h2, Y2_h2] = first_improved_method_euler(1, 2, 0.05, 1, 1);
77     print(calc_exact_y, "\n\nFirst improved method Euler:", X2_h1, Y2_h1, Y2_h2, 2);

```

```

78     auto [X3_h1, Y3_h1] = method_euler_recalculation(1, 2, 0.1, 1, 1);
79     auto [X3_h2, Y3_h2] = method_euler_recalculation(1, 2, 0.05, 1, 1);
80     print(calc_exact_y, "\n\nFirst method Euler Recalculation:", X3_h1, Y3_h1, Y2_h2,
81           2);
81     return 0;
82 }

1  #include "./matrix.h"
2  #include "./lab4_1.h"
3
4  vector<double> f(double x, double y, double z) {
5      vector<double> F(2);
6      F[0] = z;
7      F[1] = ((2*x+1)*y + 4*x*z)/4;
8      return F;
9  }
10
11 double calc_exact_y(double x) {
12     return 3*x + exp(-2*x*x);
13 }
14
15 double phi(double x_start, double x_finish, double h, double y0, double n, double y1)
16 {
17     vector<vector<double>> res = method_runge_kutta_4(f, x_start, x_finish, h, y0, n);
18     return res[1].back() - y1;
19 }
20
21 vector<vector<double>> shooting(double x_start, double x_finish, double h, double y0,
22 double y1, double n0, double n1, double eps) {
23     double ni = n0, ni_1 = n1;
24     double phi_ni = phi(x_start, x_finish, h, y0, ni, y1);
25     double phi_ni_1 = phi(x_start, x_finish, h, y0, ni_1, y1);
26     while (abs(phi_ni_1) > eps) {
27         double ni_2 = ni_1 - (ni_1 - ni) / (phi_ni_1 - phi_ni) * phi_ni_1;
28         ni = ni_1;
29         ni_1 = ni_2;
30         phi_ni = phi_ni_1;
31         phi_ni_1 = phi(x_start, x_finish, h, y0, ni_1, y1);
32     }
33     return method_runge_kutta_4(f, x_start, x_finish, h, y0, ni_1);
34 }
35
36 double p(double x) {
37     return (x - 3) / (x * x - 1);
38 }
39
40 double q(double x) {
41     return -1 / (x * x - 1);
42 }

```

```

42 double f2(double x) {
43     return 0;
44 }
45
46 vector<vector<double>> finite_difference(double x_start, double x_finish, double y0,
47     double y1, double h) {
48     int n = (x_finish - x_start) / h + 1;
49     vector<double> X(n);
50     for (int i = 0; i < n; ++i) {
51         X[i] = x_start + i * h;
52     }
53     Matrix A(n, n), B(n, 1);
54     A(0, 0) = h;
55     A(0, 1) = 0;
56     for (int i = 1; i < n - 1; ++i) {
57         A(i, i - 1) = 1 - p(X[i]) * h / 2;
58         A(i, i) = -2 + h * h * q(X[i]);
59         A(i, i + 1) = 1 + p(X[i]) * h / 2;
60     }
61     A(n - 1, n - 2) = 0;
62     A(n - 1, n - 1) = h;
63     B(0, 0) = h * y0;
64     for (int i = 1; i < n - 1; ++i) {
65         B(i, 0) = h * h * f2(X[i]);
66     }
67     B(n - 1, 0) = h * y1;
68     Matrix C = A.run_through_method(B);
69     vector<double> Y;
70     for (int i = 0; i < C.GetRows(); ++i) {
71         Y.push_back(C(i, 0));
72     }
73     return {X, Y};
74 }
75
76 int main() {
77     double h1 = 0.1, h2 = 0.05;
78     vector<vector<double>> res = shooting(2, 3, h1, -6, 0, 5, 7, 0.000001);
79     vector<double> X = res[0];
80     vector<double> Y_h1 = res[1];
81     res = shooting(2, 3, h2, -6, 0, 5, 7, 0.000001);
82     vector<double> Y_h2 = res[1];
83     print(calc_exact_y, "Method shooting:", X, Y_h1, Y_h2, 4);
84     vector<vector<double>> res2 = finite_difference(2, 3, -6, 0, h1);
85     X = res2[0];
86     vector<double> Y2_h1 = res2[1];
87     res2 = finite_difference(2, 3, -6, 0, h2);
88     Y_h2 = res2[1];
89     print(calc_exact_y, "\nFinite difference method:", X, Y2_h1, Y_h2, 2);
90     vector<double> exact_y;

```

```
90 ||   for (int i = 0; i < X.size(); ++i) {  
91 ||       exact_y.push_back(calc_exact_y(X[i]));  
92 ||   }  
93 ||   return 0;  
94 || }
```