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:	

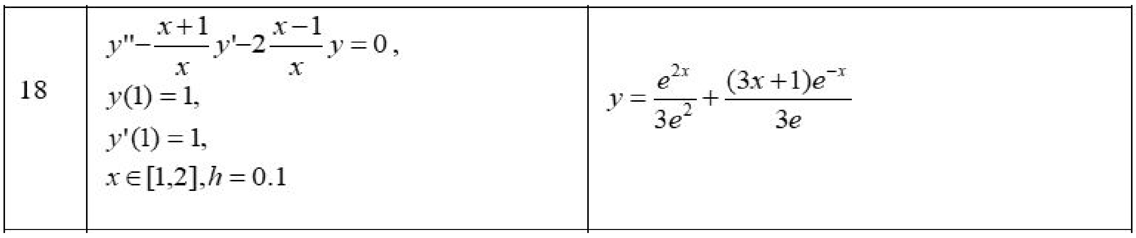
Москва, 2024

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.

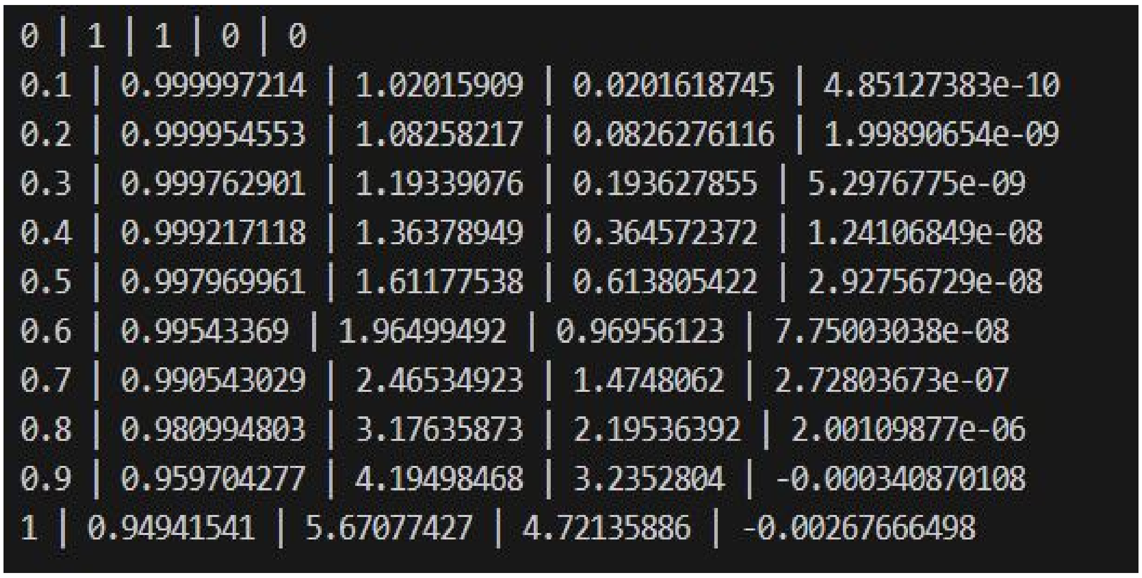
Вариант: 18



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

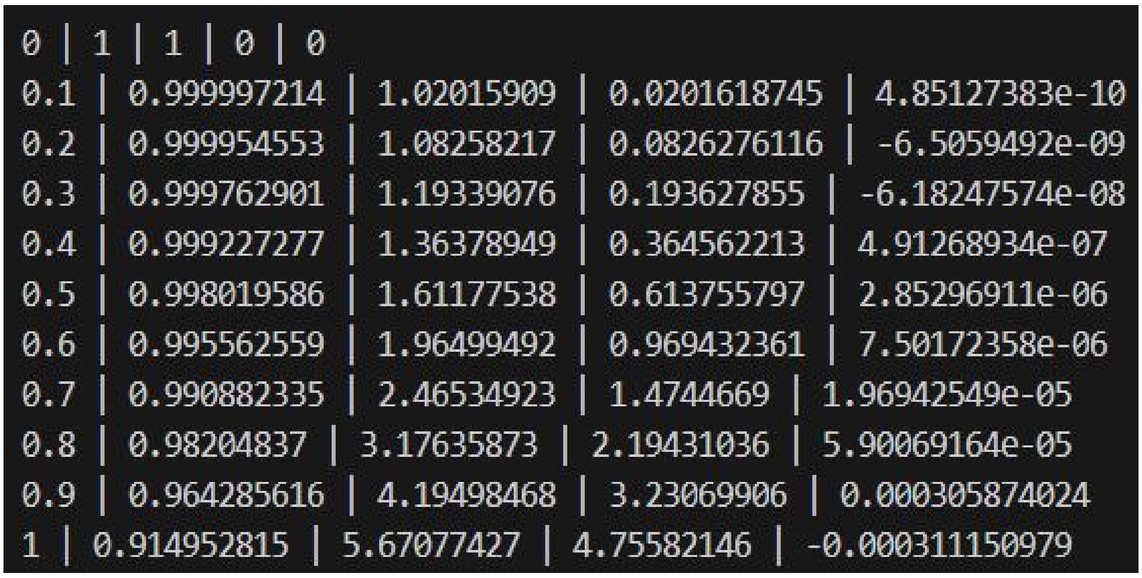
Рис. 1: Входные данные

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
```

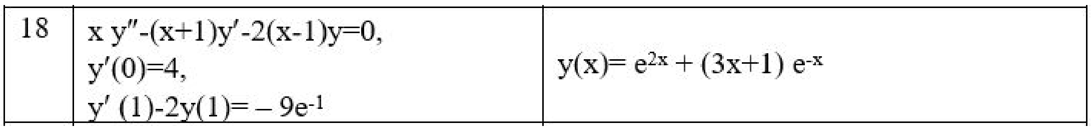
Рис. 3: Вывод программы в консоли

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.

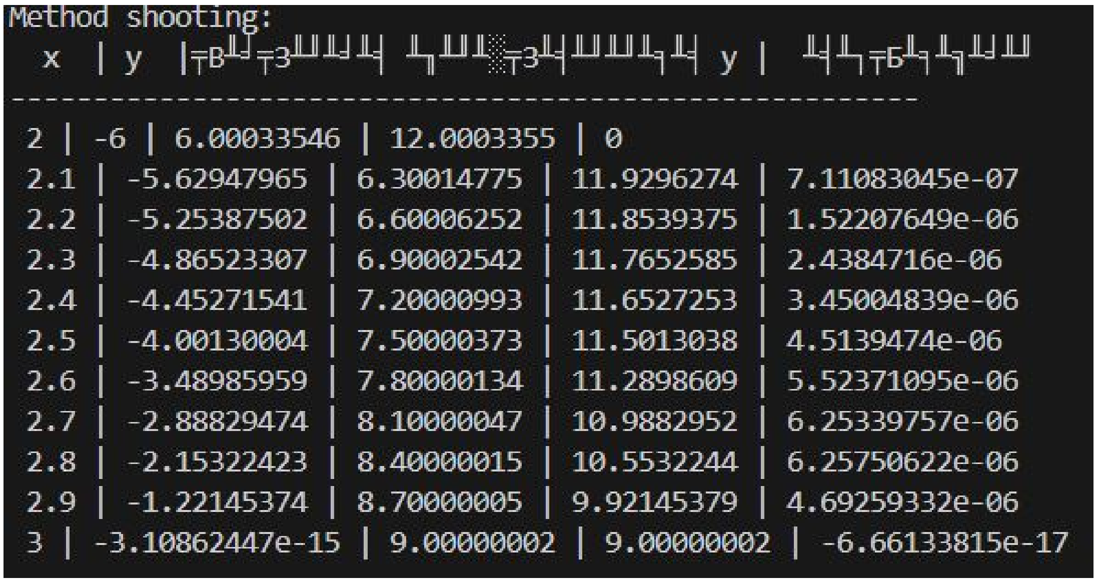
Вариант: 18



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

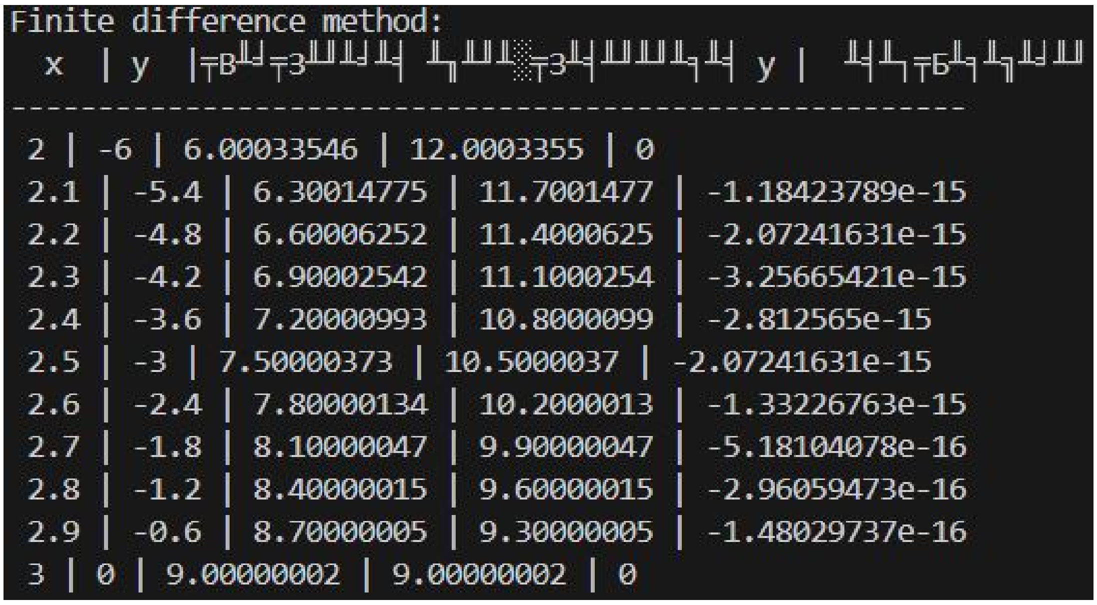
Рис. 4: Входные данные

4 Результаты работы



```
Method shooting:
  x  | y  |╤В╨╛╤З╨╜╨╛╨╡ ╨╖╨╜╨░╤З╨╡╨╜╨╕╨╡ y |  ╨╛╤И╨╕╨▒╨║╨░
-------------------------------------------------------------
 2 | -6 | 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  |╤В╨╛╤З╨╜╨╛╨╡ ╨╖╨╜╨░╤З╨╡╨╜╨╕╨╡ y |  ╨╛╤И╨╕╨▒╨║╨░
-----------------------------------------------------------------
 2 | -6 | 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

```cpp
#if !defined(MATRIX)
#define MATRIX

#include <ccomplex>
#include <cmath>
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

#define EPS 1e-5

class Matrix {
private:
    int rows_, cols_;
    vector<vector<double>> matrix_;
    vector<int> swp_;

    void SwapMatrix(Matrix &other) {
        swap(rows_, other.rows_);
        swap(cols_, other.cols_);
        swap(matrix_, other.matrix_);
    }
    Matrix Minor(int i, int j) const {
        Matrix result(rows_ - 1, cols_ - 1);
        int ki = 0;
        for (int new_i = 0; new_i < result.rows_; ++new_i) {
            int kj = 0;
            if (new_i == i)
                ki = 1;
            for (int new_j = 0; new_j < result.cols_; ++new_j) {
                if (new_j == j)
                    kj = 1;
                if (new_i + ki < rows_ && new_j + kj < cols_)
                    result.matrix_[new_i][new_j] = matrix_[new_i + ki][new_j + kj];
            }
        }
        return result;
    }

public:
    Matrix(int rows, int cols) {
        if (rows < 1 || cols < 1)
            throw runtime_error(
                "         \n");
        rows_ = rows;
        cols_ = cols;
        matrix_.resize(rows_);
        for (int i = 0; i < rows_; ++i) {
            matrix_[i].resize(cols_);
        }
    }
    Matrix() : Matrix(1, 1) {}
    Matrix(const Matrix &other) : Matrix(other.rows_, other.cols_) {
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] = other.matrix_[i][j];
            }
        }
    }
    Matrix(Matrix &&other) {
        this->SwapMatrix(other);
        other.rows_ = 0;
        other.cols_ = 0;
    }

    int GetRows() const { return rows_; }
    int GetCols() const { return cols_; }
    const vector<int> &GetSwp() const { return swp_; }

    void SetRows(int rows) {
        if (rows < 1)
            throw runtime_error("    \n");
        Matrix tmp_matrix(rows, cols_);
        for (int i = 0; i < min(tmp_matrix.rows_, rows_); ++i) {
            for (int j = 0; j < cols_; ++j) {
                tmp_matrix.matrix_[i][j] = matrix_[i][j];
            }
        }
        this->SwapMatrix(tmp_matrix);
    }
    void SetCols(int cols) {
        if (cols < 1)
            throw runtime_error("    \n");
        Matrix tmp_matrix(rows_, cols);
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < min(tmp_matrix.cols_, cols_); ++j) {
                tmp_matrix.matrix_[i][j] = matrix_[i][j];
            }
        }
        this->SwapMatrix(tmp_matrix);
    }

    bool EqMatrix(const Matrix &other) const {
        bool flag = true;
        if (rows_ != other.rows_ || cols_ != other.cols_)
            flag = false;
        else {
            for (int i = 0; i < rows_; ++i) {
                for (int j = 0; j < cols_; ++j) {
                    if (fabs(matrix_[i][j] - other.matrix_[i][j]) > EPS)
                        flag = false;
                }
            }
        }
        return flag;
    }
    void SumMatrix(const Matrix &other) {
        if (rows_ != other.rows_ || cols_ != other.cols_)
            throw runtime_error("    \n");
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] += other.matrix_[i][j];
            }
        }
    }
    void SubMatrix(const Matrix &other) {
        if (rows_ != other.rows_ || cols_ != other.cols_)
            throw runtime_error("    \n");
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] -= other.matrix_[i][j];
            }
        }
    }
    void MulNumber(const double num) {
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                matrix_[i][j] *= num;
            }
        }
    }

    Matrix MulMatrixReturn(const double num) {
        Matrix tmp = *this;
        tmp.MulNumber(num);
        return tmp;
    }

    void MulMatrix(const Matrix &other) {
        if (cols_ != other.rows_)
            throw runtime_error("   \n");
        Matrix tmp(rows_, other.cols_);
        for (int i = 0; i < rows_; ++i) {
```

```
        for (int j = 0; j < other.cols_; ++j) {
            for (int k = 0; k < cols_; ++k)
                tmp.matrix_[i][j] += matrix_[i][k] * other.matrix_[k][j];
        }
    }
    this->SwapMatrix(tmp);
}

Matrix MulMatrixReturn(const Matrix &other) {
    Matrix tmp = *this;
    tmp.MulMatrix(other);
    return tmp;
}

Matrix Transpose() const {
    Matrix result(cols_, rows_);
    for (int i = 0; i < result.rows_; ++i) {
        for (int j = 0; j < result.cols_; ++j) {
            result.matrix_[i][j] = matrix_[j][i];
        }
    }
    return result;
}

pair<Matrix, Matrix> LU() {
    swp_.clear();
    int n = this->GetRows();
    Matrix U(*this);
    Matrix L(n, n);
    for (int k = 0; k < n; ++k) { // k -      ,   ,
        int index = k; // index -  max     k
        for (int i = k + 1; i < n; ++i) {
            if (abs(U(i, k)) > abs(U(index, k))) {
                index = i;
            }
        }
        swap(U(k), U(index));
        swap(L(k), L(index));
        swp_.push_back(index);
        for (int i = k + 1; i < n; ++i) {
            double m = U(i, k) / U(k, k);
            L(i, k) = m;
            for (int j = k; j < n; ++j) {
                U(i, j) -= m * U(k, j);
            }
        }
    }
    for (int i = 0; i < n; ++i) {
        L(i, i) = 1;
    }
    return {L, U};
}

Matrix Solve(Matrix &C, Matrix &L, Matrix &U) {
    Matrix B(C);
    vector<int> swp = this->GetSwp();
    for (int i = 0; i < swp.size(); ++i) {
        swap(B(i), B(swp[i]));
    }
    int n = this->GetRows();
    // LUx = b
    // Lz = b
    Matrix Z(n, 1);
    for (int i = 0; i < n; ++i) {
        Z(i, 0) = B(i, 0);
        for (int j = 0; j < i; ++j) {
            Z(i, 0) -= L(i, j) * Z(j, 0);
        }
    }
    // Ux = z
    Matrix X(n, 1);
    for (int i = n - 1; i >= 0; --i) {
        X(i, 0) = Z(i, 0);
        for (int j = i + 1; j < n; ++j) {
            X(i, 0) -= U(i, j) * X(j, 0);
        }
        X(i, 0) = X(i, 0) / U(i, i);
    }
    return X;
}

Matrix Solve(Matrix &C) {
    auto [L, U] = this->LU();
    return this->Solve(C, L, U);
}

double Determinant() {
    // detA = det(LU) = detL * detU = detU
    double result = 1;
    auto [L, U] = this->LU();
    for (int i = 0; i < rows_; ++i) {
        result *= U(i, i);
    }
    //    swap
    int sign = 0;
    vector<int> swp = this->GetSwp();
    for (int i = 0; i < swp.size(); ++i) {
        if (swp[i] != i)
```

```
                ++sign;
        }
        if (sign % 2 != 0)
            result = -result;
        return result;
    }

    Matrix InverseMatrix() {
        int n = this->GetRows();
        Matrix B(n, 1);
        Matrix result(n, n);
        for (int i = 0; i < n; ++i) {
            if (i > 0)
                B(i - 1, 0) = 0;
            B(i, 0) = 1;
            auto [L, U] = this->LU();
            Matrix res_i = this->Solve(B, L, U);
            for (int k = 0; k < n; ++k)
                result(k, i) = res_i(k, 0);
        }
        return result;
    }

    Matrix CalcComplements() const {
        Matrix result(*this);
        if (rows_ != cols_)
            throw runtime_error(
                "        "
                "\n");
        if (rows_ == 1) {
            result.matrix_[0][0] = 1;
        } else {
            for (int i = 0; i < rows_; ++i) {
                for (int j = 0; j < cols_; ++j) {
                    Matrix new_matrix = this->Minor(i, j);
                    double minor_det = new_matrix.Determinant();
                    result.matrix_[i][j] = pow(-1, i + j) * minor_det;
                }
            }
        }
        return result;
    }

    Matrix run_through_method(Matrix &B) {
        int n = this->rows_;
        vector<double> P, Q; // x_n = P_n * x_n+1 + Q_n
        P.push_back((-1) * (*this)(0, 1) / (*this)(0, 0)); // P[0] = -c1/b1
        Q.push_back(B(0, 0) / (*this)(0, 0)); // Q[0] = d1/b1
        for (int i = 1; i < n; ++i) {
```

```cpp
            if (i == n - 1) {
                P.push_back(0); // c_n = 0
            } else {
                P.push_back((-1) * (*this)(i, i + 1) / ((*this)(i, i) + (*this)(i, i - 1) * P[i - 1])); // P_i = -c_i / (b_i + a_i * P_i-1)
            }
            Q.push_back((B(i, 0) - (*this)(i, i - 1) * Q[i - 1]) / ((*this)(i, i) + (*this)(i, i - 1) * P[i - 1])); // Q_i = (d_i - a_i * Q_i-1) / (b_i + a_i * P_i-1)
        }
        Matrix X(n, 1);
        X(n - 1, 0) = Q[n - 1];
        for (int i = n - 2; i >= 0; --i) {
            X(i, 0) = P[i] * X(i + 1, 0) + Q[i];
        }
        return X;
    }

    double norm() {
        double res = 0;
        for (int i = 0; i < this->rows_; ++i) {
            double tmp_res = 0;
            for (int j = 0; j < this->cols_; ++j) {
                tmp_res += abs((*this)(i, j));
            }
            if (tmp_res > res)
                res = tmp_res;
        }
        return res;
    }

    pair<Matrix, int> simple_iterations(Matrix &B, double eps) {
        int n = this->rows_;
        Matrix Alpha(n, n);
        Matrix Beta(n, 1);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                Alpha(i, j) = (-1) * (*this)(i, j) / (*this)(i, i);
            }
            Alpha(i, i) = 0;
            Beta(i, 0) = B(i, 0) / (*this)(i, i);
        }
        Matrix X(n, 1), Prev_X(n, 1);
        int k = 1;
        Prev_X = Beta;
        X = Beta + Alpha * Prev_X;
        // eps_k = ||Alpha|| / (1 - ||Alpha||) * ||x_k - x_k-1||
        double eps_k = 0;
        double norm = Alpha.norm();
        if (norm >= 1) {
            eps_k = (X - Prev_X).norm();
        } else {
            eps_k = norm / (1 - norm) * (X - Prev_X).norm();
        }
        while (eps_k > eps) { // eps_k <= eps
            Prev_X = X;
            X = Beta + Alpha * X;
            if (norm >= 1) {
                eps_k = (X - Prev_X).norm();
            } else {
                eps_k = norm / (1 - norm) * (X - Prev_X).norm();
            }
            ++k;
        }
        return {X, k};
    }

    pair<Matrix, int> seidel(Matrix &R, double eps) {
        int n = this->rows_;
        Matrix Alpha(n, n), Beta(n, 1), E(n, n);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                Alpha(i, j) = (-1) * (*this)(i, j) / (*this)(i, i);
            }
            Alpha(i, i) = 0;
            E(i, i) = 1;
            Beta(i, 0) = R(i, 0) / (*this)(i, i);
        }
        // Alpha = B + C
        Matrix C(n, n), B(n, n);
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (j < i)
                    B(i, j) = Alpha(i, j);
                else
                    C(i, j) = Alpha(i, j);
            }
        }
        // x_k+1 = (E - B)^-1 * C * x_k + (E - B)^-1 * Beta
        Matrix X(n, 1), Prev_X(n, 1);
        int k = 1;
        Prev_X = Beta;
        Matrix Tmp_Beta = (E - B).InverseMatrix() * Beta;
        Matrix Tmp_Alpha = (E - B).InverseMatrix() * C;
        X = Tmp_Alpha * Prev_X + Tmp_Beta;
        double eps_k = 0;
        double norm = Alpha.norm();
        if (norm >= 1) {
            eps_k = (X - Prev_X).norm();
        } else {
            eps_k = C.norm() / (1 - norm) * (X - Prev_X).norm();
        }
        // eps_k = ||C|| / (1 - ||Alpha||) * ||x_k - x_k-1||
        while (eps_k > eps) {
            Prev_X = X;
            X = Tmp_Alpha * Prev_X + Tmp_Beta;
            if (norm >= 1) {
                eps_k = (X - Prev_X).norm();
            } else {
                eps_k = C.norm() / (1 - norm) * (X - Prev_X).norm();
            }
            ++k;
        }
        return {X, k};
    }

    // helper
    double sum_square() {
        int n = this->rows_;
        double res = 0;
        //
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (i != j)
                    res += (*this)(i, j) * (*this)(i, j);
            }
        }
        return sqrt(res);
    }

    pair<pair<Matrix, Matrix>, int> jacobi_method(double eps) {
        int n = this->rows_;
        int k = 0;
        pair<int, int> max_index;
        Matrix A = *this;
        Matrix Self_Vectors(n, n);
        for (int i = 0; i < n; ++i) {
            Self_Vectors(i, i) = 1;
        }
        while (A.sum_square() > eps) {
            Matrix U(n, n);
            max_index = {1, 0}; // index abs(max_elem)
            for (int i = 0; i < n; ++i) {
                for (int j = 0; j < n; ++j) {
                    if (i != j && abs(A(i, j)) > abs(A(max_index.first, max_index.second
                        )))
                        max_index = {i, j};
                }
            }
            for (int i = 0; i < n; ++i) {
                U(i, i) = 1;
            }
            // phi = 1/2 * arctg (2 * a(i, j) / (a(i, i) - a(j, j)))
            // phi = PI/4, a(i, i) = a(j, j)
            double phi;
            if (A(max_index.first, max_index.first) == A(max_index.second, max_index.
                second))
                phi = M_PI / 4;
            else
                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)));
            U(max_index.first, max_index.first) = cos(phi);
            U(max_index.first, max_index.second) = (-1) * sin(phi);
            U(max_index.second, max_index.first) = sin(phi);
            U(max_index.second, max_index.second) = cos(phi);
            Matrix U_T = U.Transpose();
            // A^k+1 = U_T^k * A^k * U^k
            A = U_T.MulMatrixReturn(A).MulMatrixReturn(U);
            Self_Vectors.MulMatrix(U);
            ++k;
        }
        return {{A, Self_Vectors}, k};
    }

    int sign(double x) {
        if (x > 0)
            return 1;
        if (x < 0)
            return -1;
        return 0;
    }

    pair<Matrix, Matrix> qr_decomposition() {
        int n = this->rows_;
        Matrix E(n, n);
        for (int i = 0; i < n; ++i) {
            E(i, i) = 1;
        }
        Matrix Q = E;
        Matrix A = *this;
        for (int i = 0; i < n - 1; ++i) {
            Matrix H(n, n);
            Matrix V(n, 1);
            // v_1 = a_11 + sign(a11) * || 1||
            // v_i = a_i1
            double norm = 0;
```

```
            for (int j = i; j < n; ++j) {
                norm += A(j, i) * A(j, i);
            }
            norm = sqrt(norm);
            for (int j = i; j < V.GetRows(); ++j) {
                if (j == i) {
                    V(j, 0) = A(i, i) + sign(A(i, i)) * norm;
                } else {
                    V(j, 0) = A(j, i);
                }
            }
            Matrix V_T = V.Transpose();
            // H = E - 2 * v * v_t / (v_t * v)
            H = E - V.MulMatrixReturn(V_T).MulMatrixReturn(2 / (V_T.MulMatrixReturn(V))
                (0, 0));
            A = H.MulMatrixReturn(A);
            Q = Q.MulMatrixReturn(H);
        }
        // Q^-1 = Q_T
        return {Q, A};
    }

    vector<complex<double>> qr_method(double eps) {
        int n = this->rows_;
        Matrix A = *this;
        vector<complex<double>> lambda;
        vector<complex<double>> lambda_prev;
        int counter = 0;
        int iter = 50;
        while (true) {
            auto [Q, R] = A.qr_decomposition();
            A = R.MulMatrixReturn(Q);
            // cout << "A\n";
            // A.ShowMatrix();
            if (counter != iter) {
                ++counter;
                continue;
            }
            for (int i = 0; i < n; i += 1) {
                double sum = 0;
                for (int j = i + 1; j < n; ++j) {
                    sum += abs(A(j, i));
                }
                if (sum < 0.001) {
                    lambda.push_back(A(i, i));
                } else {
                    // (a_jj - Lambda)(a_j+1,j+1 - Lambda) = aj,j+1 * aj+1, j
                    double a = 1;
                    double b = (-1) * (A(i, i) + A(i + 1, i + 1));
                    double c = A(i, i) * A(i + 1, i + 1) - A(i, i + 1) * A(i + 1, i);
                    double d = b * b - 4 * c;
                    complex<double> x1, x2;
                    if (d < 0) {
                        x1 = (-b + sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
                        x2 = (-b - sqrt((abs(d))) * complex<double>(0, 1)) / (2 * a);
                    } else {
                        x1 = (-b + sqrt(d)) / (2 * a);
                        x2 = (-b - sqrt(d)) / (2 * a);
                    }
                    lambda.push_back(x1);
                    lambda.push_back(x2);
                    ++i;
                }
            }
            bool exit = true;
            //
            if (lambda_prev.size() != 0) {
                for (int i = 0; i < lambda.size(); i++) {
                    if (abs(lambda[i] - lambda_prev[i]) > eps) {
                        exit = false;
                        break;
                    }
                }
                if (exit == true)
                    break;
            }
            lambda_prev = lambda;
            lambda.clear();
            counter = 0;
        }
        return lambda;
    }

    Matrix operator+(const Matrix &other) {
        Matrix result(*this);
        result.SumMatrix(other);
        return result;
    }
    Matrix operator-(const Matrix &other) {
        Matrix result(*this);
        result.SubMatrix(other);
        return result;
    }
    Matrix operator*(const Matrix &other) {
        Matrix result(*this);
        result.MulMatrix(other);
        return result;
    }
    Matrix operator*(const double num) {
        Matrix result(*this);
        result.MulNumber(num);
        return result;
    }
    bool operator==(const Matrix &other) { return this->EqMatrix(other); }
    Matrix operator=(const Matrix &other) {
        if (this != &other) { // b = b
            Matrix tmp(other);
            this->SwapMatrix(tmp);
        }
        return *this;
    }
    void operator+=(const Matrix &other) { this->SumMatrix(other); }
    void operator-=(const Matrix &other) { this->SubMatrix(other); }
    void operator*=(const Matrix &other) { this->MulMatrix(other); }
    void operator*=(const double num) { this->MulNumber(num); }
    double &operator()(int i, int j) {
        if (i < 0 || i >= rows_ || j < 0 || j >= cols_)
            throw runtime_error("    \n");
        return matrix_[i][j];
    }
    vector<double> &operator()(int row) { return matrix_[row]; }

    void ShowMatrix() const {
        for (int i = 0; i < rows_; ++i) {
            for (int j = 0; j < cols_; ++j) {
                cout << matrix_[i][j] << " ";
            }
            cout << "\n";
        }
    }
};

ostream &operator<<(ostream &stream, Matrix A) {
    for (int i = 0; i < A.GetRows(); i++) {
        for (int j = 0; j < A.GetCols(); j++)
            stream << A(i, j) << ' ';
        stream << '\n';
    }
    return stream;
}

istream &operator>>(istream &stream, Matrix &A) {
    for (int i = 0; i < A.GetRows(); i++) {
        for (int j = 0; j < A.GetCols(); j++)
            stream >> A(i, j);
    }
    return stream;
```

```
}

#endif // MATRIX
```

```
#include <bits/stdc++.h>

using namespace std;

double method_runge_romberg(double y1, double y2, int64_t p) {
    return (y1 - y2) / (pow(2, p) - 1);
}

vector<double> num_vector(vector<double> a, double n) {
    for (int i = 0; i < a.size(); ++i) {
        a[i] *= n;
    }
    return a;
}

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)
     {
    int n;
    if (iter == -1) {
        n = (x_finish - x_start) / h;
    } else
        n = iter;
    vector<double> X(n + 1), Y(n + 1), Z(n + 1);
    for (int i = 0; i <= n; ++i) {
        X[i] = x_start + i * h;
    }
    Y[0] = y0;
    Z[0] = z0;
    for (int i = 1; i <= n; ++i) {
        vector<double> K_1 = num_vector(f(X[i - 1], Y[i - 1], Z[i - 1]), h);
        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);
        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);
        vector<double> K_4 = num_vector(f(X[i - 1] + h, Y[i - 1] + K_3[0], Z[i - 1] +
            K_3[1]), h);
        Y[i] = Y[i - 1] + 1.0 / 6 * (K_1[0] + 2 * K_2[0] + 2 * K_3[0] + K_4[0]);
        Z[i] = Z[i - 1] + 1.0 / 6 * (K_1[1] + 2 * K_2[1] + 2 * K_3[1] + K_4[1]);
    }
    return {X, Y, Z};
}

void print(double (*calc_exact_y)(double), string method, vector<double> &X_h1, vector
    <double> &Y_h1, vector<double> &Y_h2, int64_t p) {
    cout << method << "\n"
```

```cpp
        << " x |" << " y |" << "  y |" << "\t\t | -\n-------------------------------------------------------\n";
    for (int i = 0; i < X_h1.size(); ++i) {
        double exact_y = calc_exact_y(X_h1[i]);
        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;
    }
}
```

```cpp
#include "./lab4_1.h"

vector<double> f(double x, double y, double z) {
    vector<double> F(2);
    F[0] = z;
    F[1] = (exp(x*x) + 4*x*z - y)/(4*x*x-3);
    return F;
}

double calc_exact_y(double x) {
    return (exp(x) + exp(-x) - 1) * exp(x*x);
}

pair<vector<double>, vector<double>> method_adams(double x_start, double x_finish, double h, double y0, double z0) {
    int n = (x_finish - x_start) / h;
    vector<double> X(n + 1), Y(n + 1), Z(n + 1);
    for (int i = 0; i <= n; ++i) {
        X[i] = x_start + i * h;
    }
    vector<vector<double>> res_runge_kutta = method_runge_kutta_4(f, x_start, x_finish, h, y0, z0, 4);
    Y[0] = res_runge_kutta[1][0];
    Y[1] = res_runge_kutta[1][1];
    Y[2] = res_runge_kutta[1][2];
    Y[3] = res_runge_kutta[1][3];
    Z[0] = res_runge_kutta[2][0];
    Z[1] = res_runge_kutta[2][1];
    Z[2] = res_runge_kutta[2][2];
    Z[3] = res_runge_kutta[2][3];
    for (int i = 4; i <= n; ++i) {
        vector<double> F_k = f(X[i - 1], Y[i - 1], Z[i - 1]);
        vector<double> F_k_1 = f(X[i - 2], Y[i - 2], Z[i - 2]);
        vector<double> F_k_2 = f(X[i - 3], Y[i - 3], Z[i - 3]);
        vector<double> F_k_3 = f(X[i - 4], Y[i - 4], Z[i - 4]);
        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]);
        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]);
    }
    return {X, Y};
}

int main() {
    double start_pos = 0, end_pos = 1, h = 0.1;
    vector<vector<double>> res_h1 = method_runge_kutta_4(f, start_pos, end_pos, h, 1,
        0);
    vector<double> X = res_h1[0];
    vector<double> Y_h1 = res_h1[1];
    vector<vector<double>> res_h2 = method_runge_kutta_4(f, start_pos, end_pos, h/2, 1,
         0);
    vector<double> Y_h2 = res_h2[1];
    print(calc_exact_y, " -:", X, Y_h1, Y_h2, 4);
    auto [X2_h1, Y2_h1] = method_adams(start_pos, end_pos, h, 1, 0);
    auto [X2_h2, Y2_h2] = method_adams(start_pos, end_pos, h/2, 1, 0);
    print(calc_exact_y, "\n :", X2_h1, Y2_h1, Y2_h2, 4);
    return 0;
}
```

```
#include "./lab4_1.h"

vector<double> f(double x, double y, double z) {
    vector<double> F(2);
    F[0] = z;
    F[1] = (exp(x*x) + 4*x*z - y)/(4*x*x-3);
    return F;
}

double calc_exact_y(double x) {
    return (exp(x) + exp(-x) - 1) * exp(x*x);
}

pair<vector<double>, vector<double>> method_euler(double x_start, double x_finish,
    double h, double y0, double z0) {
    int n = (x_finish - x_start) / h;
    vector<double> X(n + 1), Y(n + 1), Z(n + 1);
    for (int i = 0; i <= n; ++i) {
        X[i] = x_start + i * h;
    }
    Y[0] = y0;
    Z[0] = z0;
    for (int i = 1; i <= n; ++i) {
        vector<double> F = f(X[i - 1], Y[i - 1], Z[i - 1]);
        Y[i] = Y[i - 1] + h * F[0];
        Z[i] = Z[i - 1] + h * F[1];
    }
    return {X, Y};
}

pair<vector<double>, vector<double>> first_improved_method_euler(double x_start,
    double x_finish, double h, double y0, double z0) {
    h = h / 2;
    int n = (x_finish - x_start) / h;
    vector<double> X(n + 1), Y(n + 1), Z(n + 1);
    for (int i = 0; i <= n; ++i) {
        X[i] = x_start + i * h;
    }
    Y[0] = y0;
    Z[0] = z0;
    for (int i = 1; i <= n; ++i) {
        vector<double> F = f(X[i - 1], Y[i - 1], Z[i - 1]);
        if (i % 2 == 0) {
            Y[i] = Y[i - 2] + h * 2 * F[0];
            Z[i] = Z[i - 2] + h * 2 * F[1];
        } else {
            Y[i] = Y[i - 1] + h * F[0];
            Z[i] = Z[i - 1] + h * F[1];
        }
    }
    return {X, Y};
}

pair<vector<double>, vector<double>> method_euler_recalculation(double x_start, double
     x_finish, double h, double y0, double z0) {
    int n = (x_finish - x_start) / h;
    vector<double> X(n + 1), Y(n + 1), Y_tmp(n + 1), Z(n + 1), Z_tmp(n + 1);
    for (int i = 0; i <= n; ++i) {
        X[i] = x_start + i * h;
    }
    Y[0] = y0;
    Z[0] = z0;
    for (int i = 1; i <= n; ++i) {
        vector<double> F_tmp = f(X[i - 1], Y[i - 1], Z[i - 1]);
        Y_tmp[i] = Y[i - 1] + h * F_tmp[0];
        Z_tmp[i] = Z[i - 1] + h * F_tmp[1];
        vector<double> F = f(X[i], Y_tmp[i], Z_tmp[i]);
        Y[i] = Y[i - 1] + h * (F_tmp[0] + F[0]) / 2;
        Z[i] = Z[i - 1] + h * (F_tmp[1] + F[1]) / 2;
    }
    return {X, Y};
}

int main() {
    auto [X_h1, Y_h1] = method_euler(1, 2, 0.1, 1, 1);
    auto [X_h2, Y_h2] = method_euler(1, 2, 0.05, 1, 1);
    print(calc_exact_y, "Method Euler:", X_h1, Y_h1, Y_h2, 1);
    auto [X2_h1, Y2_h1] = first_improved_method_euler(1, 2, 0.1, 1, 1);
    auto [X2_h2, Y2_h2] = first_improved_method_euler(1, 2, 0.05, 1, 1);
    print(calc_exact_y, "\n\nFirst improved method Euler:", X2_h1, Y2_h1, Y2_h2, 2);
    auto [X3_h1, Y3_h1] = method_euler_recalculation(1, 2, 0.1, 1, 1);
    auto [X3_h2, Y3_h2] = method_euler_recalculation(1, 2, 0.05, 1, 1);
    print(calc_exact_y, "\n\nFirst method Euler Recalculation:", X3_h1, Y3_h1, Y2_h2,
        2);
    return 0;
}
```

```
#include "./matrix.h"
#include "./lab4_1.h"

vector<double> f(double x, double y, double z) {
    vector<double> F(2);
    F[0] = z;
    F[1] = ((2*x+1)*y + 4*x*z)/4;
    return F;
}

double calc_exact_y(double x) {
    return 3*x + exp(-2*x*x);
}

double phi(double x_start, double x_finish, double h, double y0, double n, double y1)
    {
    vector<vector<double>> res = method_runge_kutta_4(f, x_start, x_finish, h, y0, n);
    return res[1].back() - y1;
}

vector<vector<double>> shooting(double x_start, double x_finish, double h, double y0,
    double y1, double n0, double n1, double eps) {
    double ni = n0, ni_1 = n1;
    double phi_ni = phi(x_start, x_finish, h, y0, ni, y1);
    double phi_ni_1 = phi(x_start, x_finish, h, y0, ni_1, y1);
    while (abs(phi_ni_1) > eps) {
        double ni_2 = ni_1 - (ni_1 - ni) / (phi_ni_1 - phi_ni) * phi_ni_1;
        ni = ni_1;
        ni_1 = ni_2;
        phi_ni = phi_ni_1;
        phi_ni_1 = phi(x_start, x_finish, h, y0, ni_1, y1);
    }
    return method_runge_kutta_4(f, x_start, x_finish, h, y0, ni_1);
}

double p(double x) {
    return (x - 3) / (x * x - 1);
}

double q(double x) {
    return -1 / (x * x - 1);
}

```

```
double f2(double x) {
    return 0;
}

vector<vector<double>> finite_difference(double x_start, double x_finish, double y0,
    double y1, double h) {
    int n = (x_finish - x_start) / h + 1;
    vector<double> X(n);
    for (int i = 0; i < n; ++i) {
        X[i] = x_start + i * h;
    }
    Matrix A(n, n), B(n, 1);
    A(0, 0) = h;
    A(0, 1) = 0;
    for (int i = 1; i < n - 1; ++i) {
        A(i, i - 1) = 1 - p(X[i]) * h / 2;
        A(i, i) = -2 + h * h * q(X[i]);
        A(i, i + 1) = 1 + p(X[i]) * h / 2;
    }
    A(n - 1, n - 2) = 0;
    A(n - 1, n - 1) = h;
    B(0, 0) = h * y0;
    for (int i = 1; i < n - 1; ++i) {
        B(i, 0) = h * h * f2(X[i]);
    }
    B(n - 1, 0) = h * y1;
    Matrix C = A.run_through_method(B);
    vector<double> Y;
    for (int i = 0; i < C.GetRows(); ++i) {
        Y.push_back(C(i, 0));
    }
    return {X, Y};
}

int main() {
    double h1 = 0.1, h2 = 0.05;
    vector<vector<double>> res = shooting(2, 3, h1, -6, 0, 5, 7, 0.000001);
    vector<double> X = res[0];
    vector<double> Y_h1 = res[1];
    res = shooting(2, 3, h2, -6, 0, 5, 7, 0.000001);
    vector<double> Y_h2 = res[1];
    print(calc_exact_y, "Method shooting:", X, Y_h1, Y_h2, 4);
    vector<vector<double>> res2 = finite_difference(2, 3, -6, 0, h1);
    X = res2[0];
    vector<double> Y2_h1 = res2[1];
    res2 = finite_difference(2, 3, -6, 0, h2);
    Y_h2 = res2[1];
    print(calc_exact_y, "\nFinite difference method:", X, Y2_h1, Y_h2, 2);
    vector<double> exact_y;
    for (int i = 0; i < X.size(); ++i) {
        exact_y.push_back(calc_exact_y(X[i]));
    }
    return 0;
}
```