Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет безопасности информационных технологий

Дисциплина:

«Алгоритмы и структуры данных»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

«Решение алгебраического уравнения Риккати с сингулярной матрицей методом ADI»

Выполнили:

Дынина Е.А., студент группы N3249

_____
(подпись)

Проверил:

Ерофеев С.А.

_____
(отметка о выполнении)

_____
(подпись)

Санкт-Петербург

2025 г.

СОДЕРЖАНИЕ

ПОСТАНОВКА ЗАДАЧИ

Цель работы — разработать программ, которая решает алгебраическое уравнение Риккати с сингулярной матрицей с помощью alternating direction implicit method (метод ADI).

Для достижения поставленной цели необходимо решить следующие задачи:

- Изучить алгебраическое уравнение Риккати, метод ADI и эвристический алгоритм подбора параметров для метода с помощью итераций Арнольда;
- Создать блок-схему алгоритма;
- Написать программу на языке C++ с интерфейсом для пользователей;
- Протестировать программу на данных разного размера.

ТЕХНИЧЕСКАЯ ЗАДАЧА

Алгебраическое уравнение Риккати выводится из системы, характеризующей изменение системы

$$\begin{cases} E\dot{x}(t) = Ax(t) + Bu(t), \\ y(t) = Cx(t) \end{cases}$$

В результате преобразований получаем алгебраическое уравнение Риккати

$$\mathrm{E}^T\dot{P}E + E^TPE + A^TPE + Q - E^TPBR^{-1}B^TPE = 0$$

После дискретизации по времени получаем уравнение, на котором впоследствии будем проверять сходимость результатов

$$\mathrm{E}^T(P_{k+1} - P_k)E + \tau(E^TP_{k+1}E + A^TP_{k+1}E + Q - E^TP_{k+1}BR^{-1}B^TP_{k+1}E) = 0$$

Метод ADI — численный итерационный метод, применяемый для решения алгебраических уравнений. Основная идея заключается в разделении операторов, то есть пошаговом решении уравнения. После применения метода к исходному уравнению и нескольких упрощений получаем три варианта шагов(отличие в последнем члене второго шага). Впоследствии проверим какой из них более точный.

1.

$$(E^T + \alpha\tau A^T)P_{k+\frac{1}{2}}(E + \alpha\tau A) = E^TP_kE - \tau Q + \tau E^TP_kBR^{-1}B^TP_kE$$

$$(E^T + \beta\tau A^T)P_{k+1}(E + \beta\tau A) = E^TP_kE - \tau Q + \tau E^TP_{k+\frac{1}{2}}BR^{-1}B^TP_{k+\frac{1}{2}}E$$

2.

$$(E^T + \alpha\tau A^T)P_{k+\frac{1}{2}}(E + \alpha\tau A) = E^TP_kE - \tau Q + \tau E^TP_kBR^{-1}B^TP_kE$$

$$(E^T + \beta\tau A^T)P_{k+1}(E + \beta\tau A) = E^TP_kE - \tau Q + \tau E^TP_k\,BR^{-1}B^TP_kE$$

3.

$$(E^T + \alpha\tau A^T)P_{k+\frac{1}{2}}(E + \alpha\tau A) = E^TP_kE - \tau Q + \tau E^TP_kBR^{-1}B^TP_kE$$

$$(E^T + \beta\tau A^T)P_{k+1}(E + \beta\tau A) = E^TP_kE - \tau Q + \tau E^TP_{k+\frac{1}{2}}BR^{-1}B^TP_kE$$

Параметры α и β в сумме равняются 1. Находим их с помощью эвристического алгоритма, который выбирает необходимое число наиболее разнесенных значений, полученных в результате итераций Арнольда.

Итерации Арнольда:

1) Выбрать вектор v_1 с единичной нормой
2) $\forall j = 1,2,\dots m$:
   - $h_{ij} = \left(Av_j, v_i\right)$ для $i = 1,2,\dots j$

- $w_j = Av_j - \sum_{i=1}^{j} h_{ij}\, v_i$
- $h_{j+1,j} = \|w_j\|$
- Если $h_{j+1,j} = 0$, то break
- $v_{j+1} = \frac{w_j}{h_{j+1,j}}$

Входные данные

На вход в программу через указанные заранее файлы подаются 4 матрицы с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Промежуточные данные

В ходе работы программы задействованы промежуточные переменные, указанные в таблице 1. Для облегчения вычислений также используются различные вспомогательные матрицы и векторы с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Таблица 1 – Промежуточные данные

Название переменной	**Тип в C++**	**Диапазон типа**	**Значение**
line	string	—	Строка для чтения из файла
val	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Переменная, в которую считываем числа из файла
t	int	$[0, 2147483647]$	Количество столбцов матрицы B
n	int	$[0, 2147483647]$	Количество строк матрицы A
tau	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Шаг времени
i,j,k	int	$[0, 2147483647]$	Счетчик в цикле
max_width	int	$[0, 2147483647]$	Счетчик для форматированного вывода
det	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Определитель матрицы
max_row	int	$[0, 2147483647]$	Переменная для запоминания ряда с наибольшим значением в столбце

sum	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Вспомогательная переменная для проведения итераций Арнольда
norm	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Норма
s	int	$[0, 2147483647]$	Счетчик сдвигов
best_idx	int	$[0, 2147483647]$	Индекс наиболее удаленного сдвига
l	int	$[0, 2147483647]$	Количество найденных сдвигов
max_min_dist	double	$[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$	Переменная для запоминания максимально отдаленного сдвига

Выходные данные

В результате работы программы в консольное приложение выводится полученная матрица решения уравнения с числами типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$) и его сходимость с предыдущим решением norm типа double (диапазон $[-1{,}7 \cdot 10^{308}, 1{,}7 \cdot 10^{308}]$).

Используемые функции

- **ifstream —** открывает файл для чтения;
- **istringstream —** считывает данные из строки;
- **numeric_limits <тип>::max —** возвращает максимальное конечное значение типа;
- **.resize(n) —** изменяет размер вектора на n;
- **swap(a,b) —** меняет местами содержимое переменных a и b:
- **iota(first,last, val)** — заполнит последовательность [first, last] значением val, инкрементируя его после каждого присваивания;
- **.close()** — закрывает файл;
- **.push_back(x) —** добавление элемента x в вектор;
- **.empty() —** проверка вектора на пустоту;
- **.size() —** выводит размер вектора;
- **.begin() —** возвращает итератор, указывающий на начало вектора;
- **.end() —** возвращает итератор, указывающий на конец вектора;
- **.front() —** выводит ссылку на первый элемент вектора;
- **.sqrt(x) —** возвращает квадрат элемента x;

- **.pow(x,n) —** возвращает элемент x в степени n;
- **setw() —** задает ширину поля вывода строк;
- **arnoldi(Matrix A, int m, Matrix V, Matrix H) —** считает итерации Арнольда для матрицы A;
- **computeADIShifts(Matrix A, int k, int m = 20) —** находит k сдвигов для метода ADI с помощью итераций Арнольда;
- **solveRiccatiADI(Matrix A, Matrix B, Matrix Q, Matrix R, Matrix E, int max_iter, double tau, double tol)** — решает уравнение Риккати с max_iter итерациями и шагов времени tau.

Функции класса Matrix

- **.getRows() —** возвращает количество строк в матрице;
- **.getCols() —** возвращает количество столбцов в матрице;
- **Оператор + —** сложение матриц одинаковой размерности;
- **Оператор - —** вычитание матриц одинаковой размерности;
- **Оператор * —** умножение матрицу на матрицу и матрицу на скаляр;
- **Оператор << —** вывод матрицы;
- **.transpose() —** транспонирование матрицы;
- **.readFromFile(string filename) —** читает матрицу из файла;
- **.determinant() —** возвращает определитель матрицы;
- **.inverse() —** возвращает обратную матрицу.

БЛОК-СХЕМА

Рассмотрим блок-схему придуманного алгоритма, где реализована решение уравнения Риккати методом ADI и подбор параметров эвристическим алгоритмом.

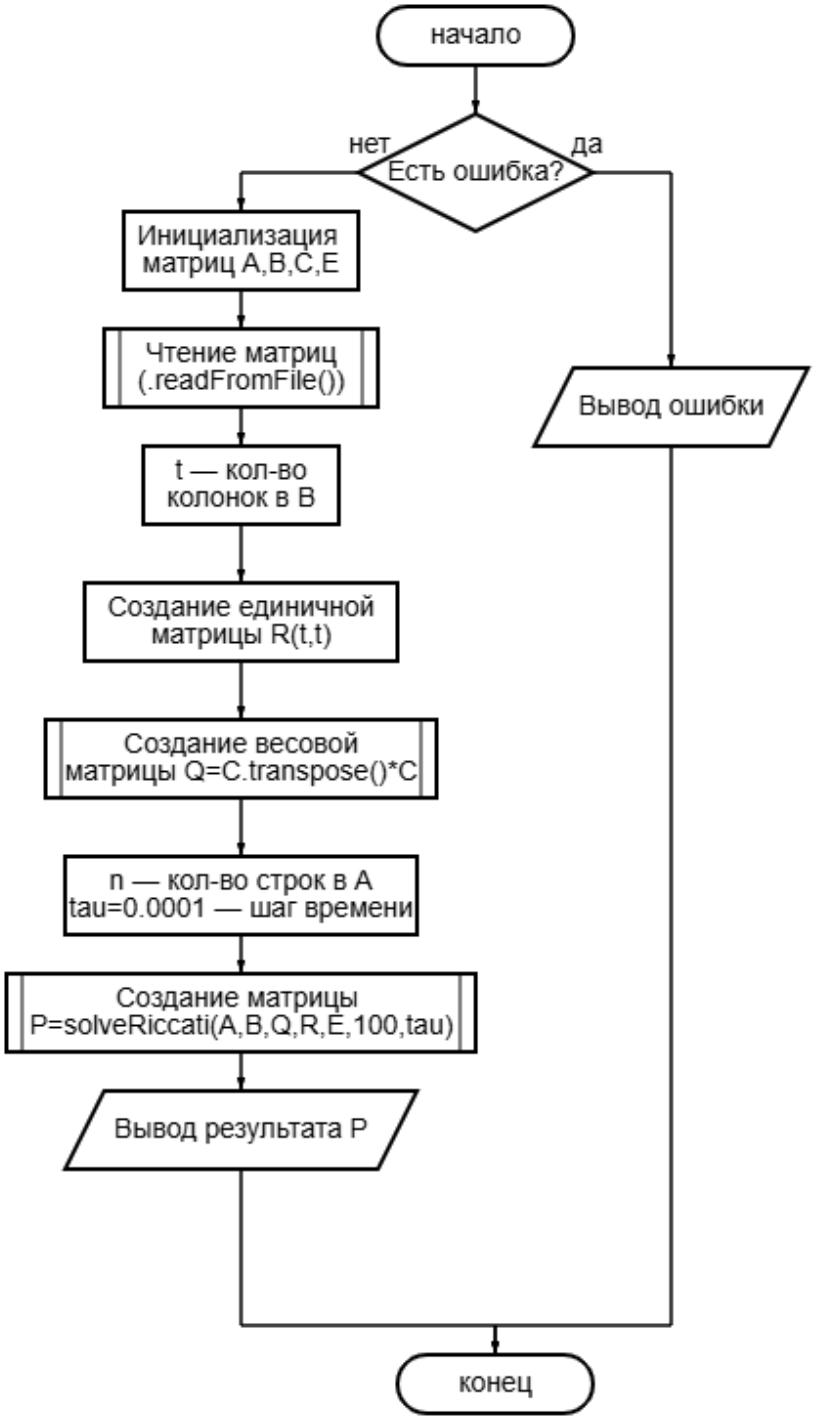

Рисунок 1 – Блок-схема основной части программы

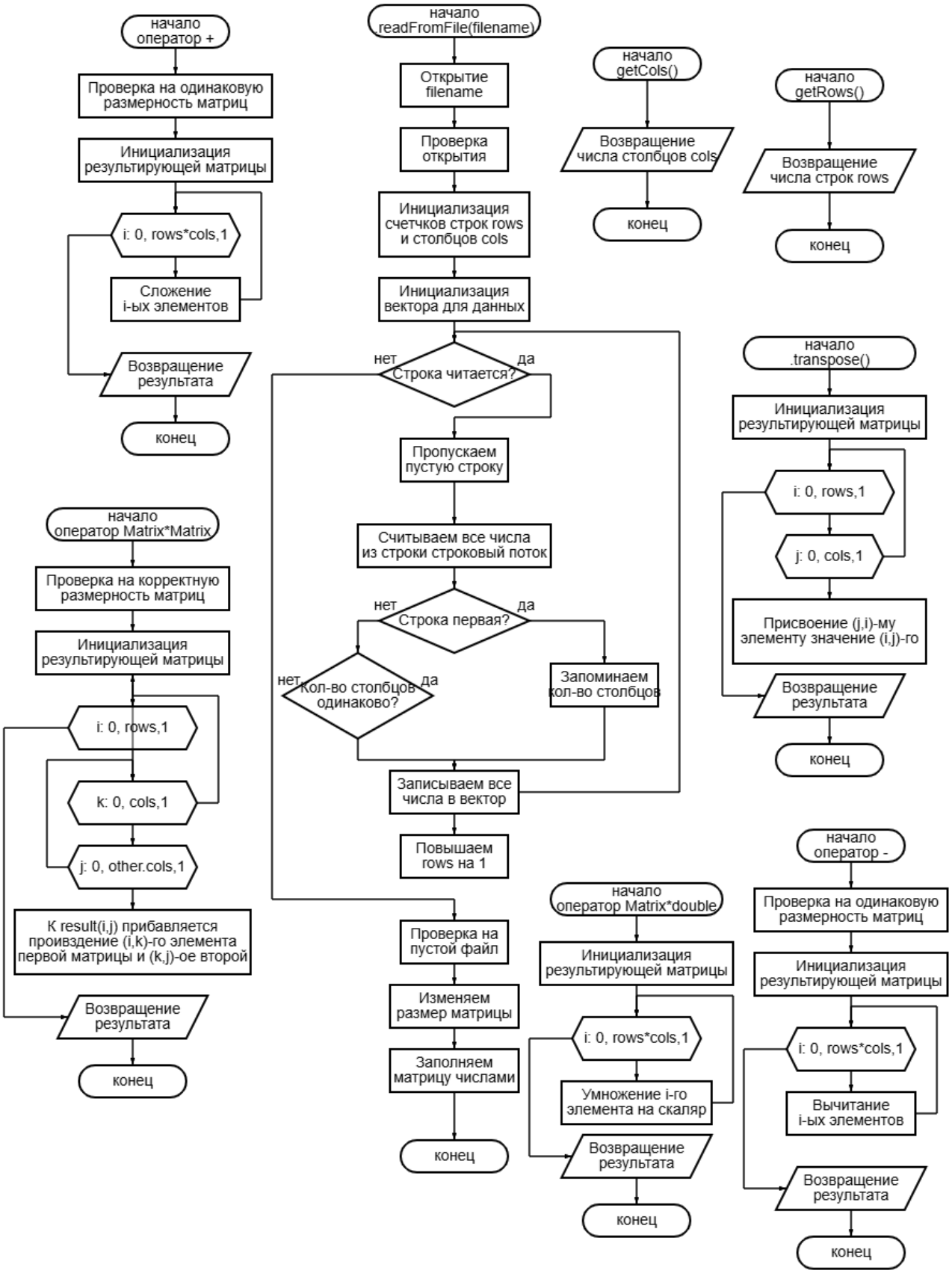

Рисунок 2 – Блок-схема функций класса Matrix

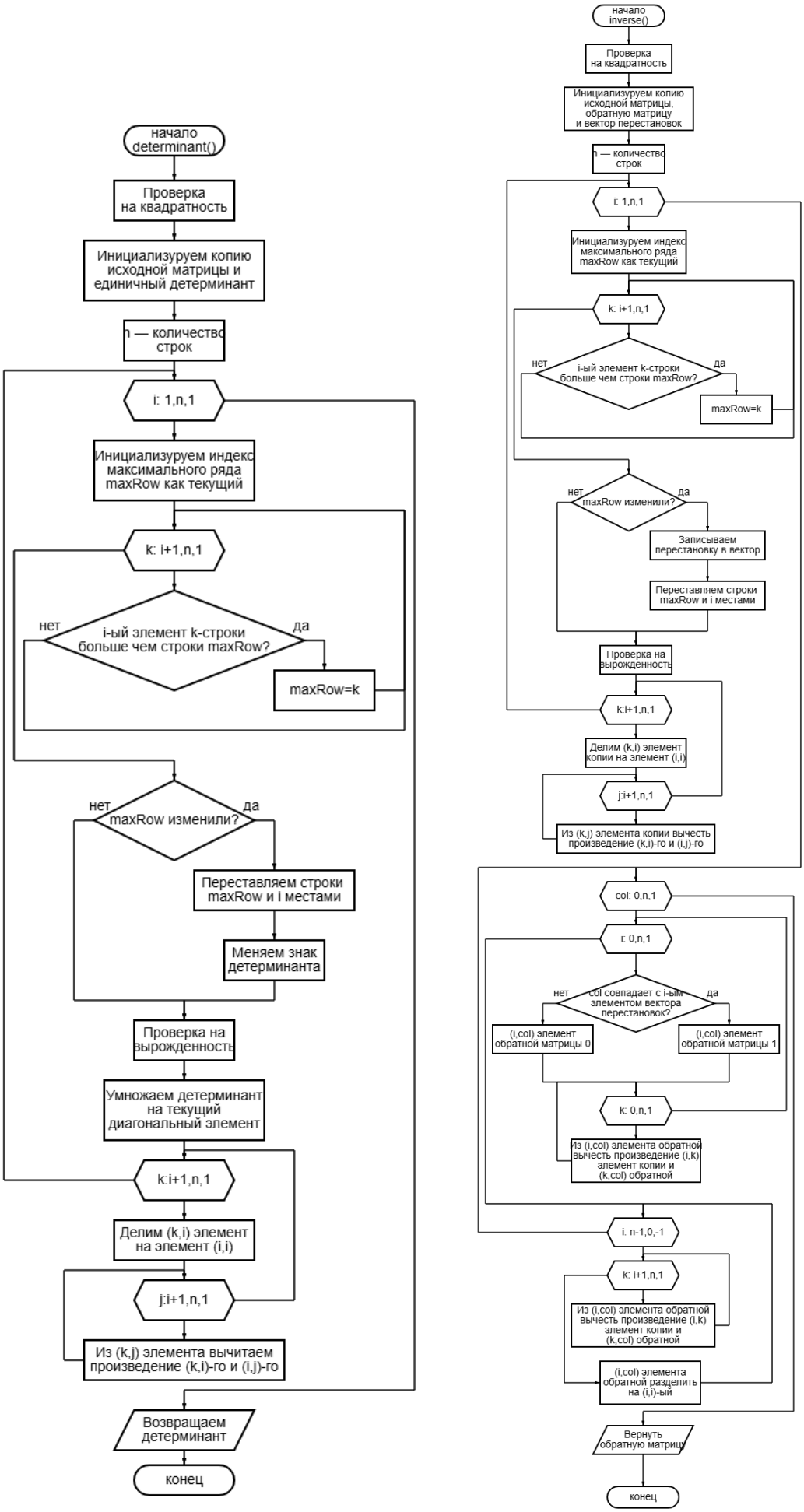

Рисунок 3 – Блок-схема функций определителя матрицы и обратной

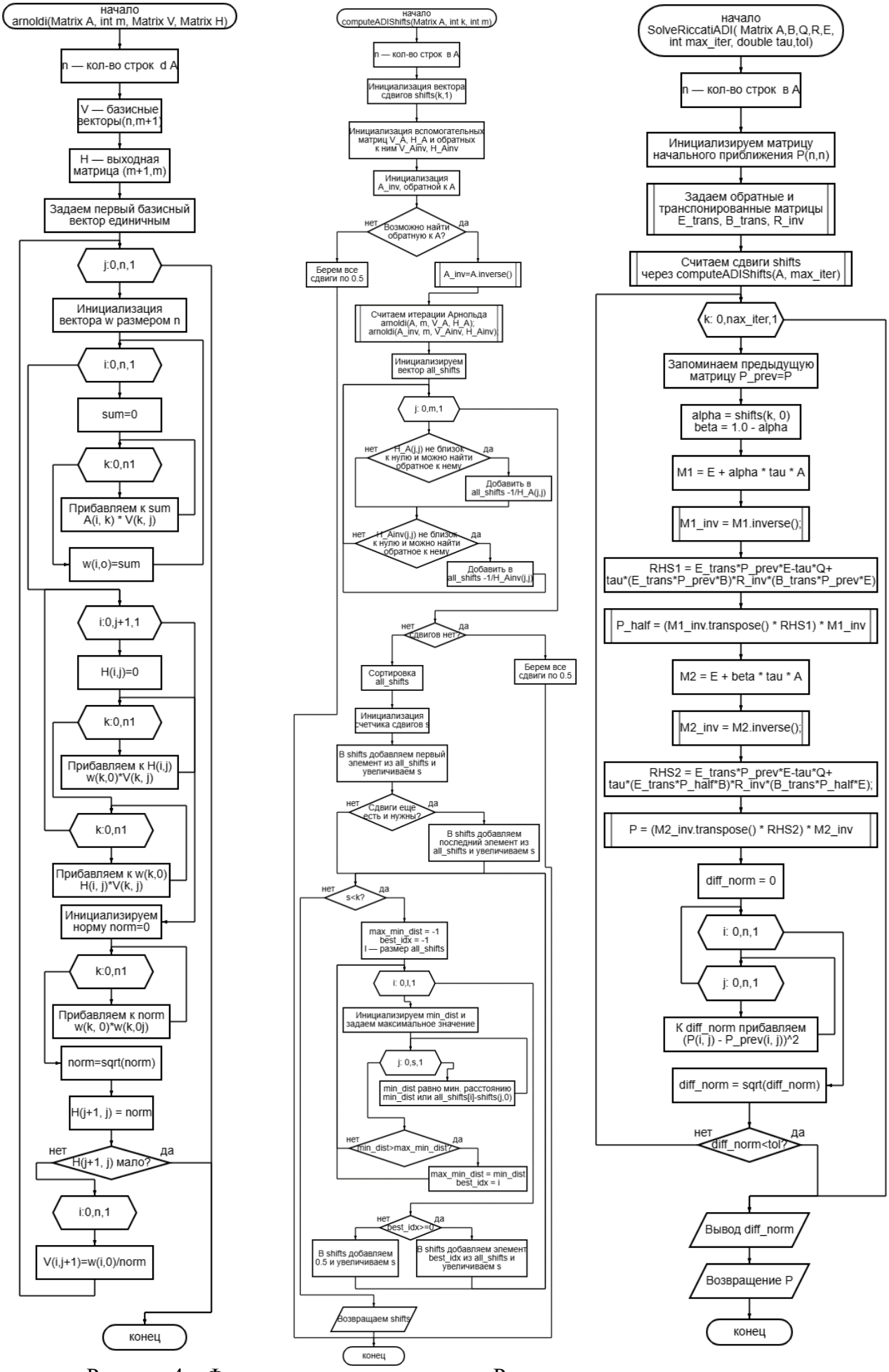

Рисунок 4 – Функции решения уравнения Риккати и нахождения сдвигов

КОД

Ниже приведен код, написанный для решения алгебраического уравнения Риккати методом ADI с реализованным классом матриц и эвристическим алгоритмом вычисления сдвигов, по указанной выше блок-схеме. Программа реализована на языке C++.

```
#include <iostream>
#include <fstream>
#include <cmath>
#include <stdexcept>
#include <iomanip>
#include <algorithm>
#include <limits>
#include <cstdlib>
#include <vector>
#include <numeric>
using namespace std;

class Matrix {
private:
    vector<double> data;
    int rows, cols;
public:
      // Конструкторы
    Matrix() : rows(0), cols(0) {}
    Matrix(int r, int c, double val = 0.0) : rows(r), cols(c), data(r * c,
val) {}
      // Доступ к элементам
    double& operator()(int i, int j) { return data[i * cols + j]; }
    const double& operator()(int i, int j) const { return data[i * cols + j];
}
      // Возвращение размеров
    int getRows() const { return rows; }
    int getCols() const { return cols; }
      // Сложение двух матриц
    Matrix operator+(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols) // Проверка на
одинаковую размерность матриц
            throw invalid_argument("Matrix dimensions do not match for
addition");
        Matrix result(rows, cols);
        for (int i = 0; i < rows * cols; ++i)
            result.data[i] = data[i] + other.data[i];
        return result;
    }
      // Вычитание матриц
    Matrix operator-(const Matrix& other) const {
        if (rows != other.rows || cols != other.cols)// Проверка на
одинаковую размерность матриц
            throw invalid_argument("Matrix dimensions do not match for
subtraction");
        Matrix result(rows, cols);
        for (int i = 0; i < rows * cols; ++i)
            result.data[i] = data[i] - other.data[i];
        return result;
    }
      // Умножение матриц
    Matrix operator*(const Matrix& other) const {
        if (cols != other.rows)// Проверка на корректную размерность матриц
```

```
            throw invalid_argument("Matrix dimensions do not match for
multiplication");
        Matrix result(rows, other.cols);
        for (int i = 0; i < rows; ++i)
            for (int k = 0; k < cols; ++k)
                for (int j = 0; j < other.cols; ++j)
                    result(i, j) += (*this)(i, k) * other(k, j);
        return result;
    }
      // Умножение на скаляр
    Matrix operator*(double scalar) const {
        Matrix result(rows, cols);
        for (int i = 0; i < rows * cols; ++i)
            result.data[i] = data[i] * scalar;
        return result;
    }
    friend Matrix operator*(double scalar, const Matrix& m) {
        return m * scalar;
    }
      // Транспонирование
    Matrix transpose() const {
        Matrix result(cols, rows);
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < cols; ++j)
                result(j, i) = (*this)(i, j);
        return result;
    }
      // Чтение матрицы из файла
    void readFromFile(const string& filename) {
        ifstream file(filename);
        if (!file) throw runtime_error("Error opening file: " + filename);
            // Инициализация счетчиков строк и столбцов
        rows = 0;
        cols = 0;
        string line;
        vector<vector<double>> tempData;
            // Читаем файл построчно
        while (getline(file, line)) {
            if (line.empty()) continue;
            istringstream iss(line); // Используем строковый поток
            vector<double> row;
            double value;
            while (iss >> value) row.push_back(value);// Читаем все числа из
строки
            if (rows == 0) cols = row.size(); // Если первая строка, то
запоминаем размер
            else if ((int)row.size() != cols) // Проверка несогласованности
количества столбцов
                throw runtime_error("Inconsistent number of columns in
matrix");
            tempData.push_back(row);
            rows++;
        }
            // Проверка на пустоту
        if (rows == 0 || cols == 0)
            throw runtime_error("Empty matrix in file");
            // Изменяем размер хранилища данных и заполняем его
        data.resize(rows * cols);
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < cols; ++j)
                (*this)(i, j) = tempData[i][j];
    }
```

```
      // Вывод матрицы в поток
    friend ostream& operator<<(ostream& os, const Matrix& m) {
        int max_width = 0;
        for (int i = 0; i < m.rows; ++i)
            for (int j = 0; j < m.cols; ++j) {
                double val = abs(m(i, j)) < 1e-10 ? 0 : m(i, j); // Обнуляем
малые числа
                ostringstream oss; // Строковый поток для определение ширины
                if (val == floor(val)) oss << (int)val; // Целые как целые
выводим
                else oss << val;
                max_width = max(max_width, (int)oss.str().length()); //
Обновляем максимальную ширину
            }
            // Выводим также матрицу с выравниванием
        for (int i = 0; i < m.rows; ++i) {
            for (int j = 0; j < m.cols; ++j) {
                double val = abs(m(i, j)) < 1e-10 ? 0 : m(i, j);
                if (val == floor(val)) os << setw(max_width + 1) << (int)val;
                else os << setw(max_width + 1) << val;
            }
            os << endl;
        }
        return os;
    }
      // Вычисление определителя
    double determinant() const {
      // Проверка на квадратность
        if (rows != cols) throw invalid_argument("Matrix must be square");
        Matrix LU(*this);
        double det = 1;
        int n = rows;
        for (int i = 0; i < n; ++i) {
            int maxRow = i;
            // Определяем строку с максимальным по модуля элементов в текущем
столбце
            for (int k = i + 1; k < n; ++k)
                if (abs(LU(k, i)) > abs(LU(maxRow, i))) maxRow = k;
            if (maxRow != i) {// Перестановка строк
                swap(LU(i, i), LU(maxRow, i));
                det *= -1;
            }
            if (abs(LU(i, i)) < 1e-12) return 0;// Проверка на вырожденность
            det *= LU(i, i); // Умножение определителя на диогональный
элемент
            for (int k = i + 1; k < n; ++k) {
                  LU(k,i) /= LU(i,i); // Сохраняем множитель
                for (int j = i + 1; j < n; ++j) {
                    LU(k, j) -= LU(k, i) * LU(i, j); //Обновляем матрицу
                }
            }
        }
        return det;
    }
      // Обратная матрица
      Matrix inverse(double tol = 1e-12) const {
          if (rows != cols)
              throw invalid_argument("Matrix must be square to compute
inverse");

          int n = rows;
          Matrix inv(n, n);  // Обратная матрица
```

```
            Matrix LU(*this);   // Копия исходной матрицы
            vector<int> perm(n); // Вектор перестановок строк
            iota(perm.begin(), perm.end(), 0); // Инициализация: [0, 1, 2, ...,
n-1]
            for (int i = 0; i < n; ++i) {
                // Поиск максимального элемента в столбце i
                int maxRow = i;
                for (int k = i + 1; k < n; ++k)
                    if (abs(LU(k, i)) > abs(LU(maxRow, i)))
                        maxRow = k;
                // Перестановка строк, если необходимо
                if (maxRow != i) {
                    swap(perm[i], perm[maxRow]);
                    for (int k = 0; k < n; ++k)
                        swap(LU(i, k), LU(maxRow, k));
                }
                // Проверка на вырожденность
                if (abs(LU(i, i)) < tol)
                    throw invalid_argument("Matrix is singular (pivotal element
< tolerance)");
                // Вычисление множителей L и обновление U
                for (int k = i + 1; k < n; ++k) {
                    LU(k, i) /= LU(i, i);  // Сохраняем множитель для L
                    for (int j = i + 1; j < n; ++j)
                        LU(k, j) -= LU(k, i) * LU(i, j);  // Обновляем U
                }
            }
            // Решение LUX = I для каждого столбца обратной матрицы
            for (int col = 0; col < n; ++col) {
                // Прямой ход: решение LY = Pb (где b - столбец единичной
матрицы)
                for (int i = 0; i < n; ++i) {
                    inv(i, col) = (perm[i] == col) ? 1 : 0;  // Учитываем
перестановки
                    for (int k = 0; k < i; ++k)
                        inv(i, col) -= LU(i, k) * inv(k, col);  // Используем
L[i][k]
                }

                // Обратный ход: решение UX = Y
                for (int i = n - 1; i >= 0; --i) {
                    for (int k = i + 1; k < n; ++k)
                        inv(i, col) -= LU(i, k) * inv(k, col);  // Используем
U[i][k]
                    inv(i, col) /= LU(i, i);  // Делим на диагональный элемент U
                }
            }

            return inv;
        }
};

// Итерации Арнольда
void arnoldi(const Matrix& A, int m, Matrix& V, Matrix& H) {
    int n = A.getRows();
    V = Matrix(n, m + 1);// Базисные векторы
    H = Matrix(m + 1, m);// Выходная матрица

    V(0, 0)=1; // Единичный первый базисный вектор
    for (int j = 0; j < m; ++j) {
      // Умножение матрицы на вектор (A*v_j)
        Matrix w(n, 1);
```

```
        for (int i = 0; i < n; ++i) {
            double sum = 0;
            for (int k = 0; k < n; ++k) sum += A(i, k) * V(k, j);
            w(i, 0) = sum;
        }
        // Вычисление H и w
        for (int i = 0; i <= j; i++) {
            H(i, j) = 0;
            for (int k = 0; k < n; ++k) H(i, j) += w(k, 0) * V(k, i);
            for (int k = 0; k < n; ++k) w(k, 0) -= H(i, j) * V(k, i);
        }
        // Нормализация
        double norm = 0;
        for (int k = 0; k < n; ++k) norm+=w(k,0)*w(k,0);
        norm=sqrt(norm);
        H(j + 1, j) = norm;
        if (H(j + 1, j) < 1e-10) break;
        for (int i = 0; i < n; ++i) V(i, j + 1) = w(i, 0) / norm;
    }
}
// Вычисление сдвигов
Matrix computeADIShifts(const Matrix& A, int k, int m = 20) {
    int n = A.getRows();
    Matrix shifts(k, 1);
    Matrix V_A, H_A, V_Ainv, H_Ainv;
    Matrix A_inv;
      // Если не удалось вычислить обратную, то берем сдвиги по 0.5
    try {
        A_inv = A.inverse();
    } catch (...) {
        for (int i = 0; i < k; ++i) shifts(i, 0) = 0.5;
        return shifts;
    }
      // Итерации Арнольда
    arnoldi(A, m, V_A, H_A);
    arnoldi(A_inv, m, V_Ainv, H_Ainv);

    vector<double> all_shifts;
    for (int j = 0; j < m; ++j) {
        if (abs(H_A(j, j)) > 1e-12 && !isnan(-1.0 / H_A(j, j)))
            all_shifts.push_back(-1.0 / H_A(j, j));
        if (abs(H_Ainv(j, j)) > 1e-12 && !isnan(-1.0 / H_Ainv(j, j)))
            all_shifts.push_back(-1.0 / H_Ainv(j, j));
    }
      // Если сдвигов нет, берем по 0.5
    if (all_shifts.empty()) {
        for (int i = 0; i < k; ++i) shifts(i, 0) = 0.5;
        return shifts;
    }

    sort(all_shifts.begin(), all_shifts.end());
    // Добавляем минимальный и максимальные сдвиги
    int s = 0;
    shifts(s++, 0) = all_shifts.front();
    if (k > 1 && all_shifts.size() > 1) shifts(s++, 0) = all_shifts.back();
      // Добавляем оставшиеся максимально разнесенные сдвиги
    while (s < k) {
        double max_min_dist = -1;
        int best_idx = -1;
        int l=all_shifts.size();
        for (int i = 0; i<l; ++i) {
            double min_dist = numeric_limits<double>::max();
```

```
            for (int j = 0; j < s; ++j) //Ищем мин. расстояние до выбранных
сдвигов
                min_dist = min(min_dist, abs(all_shifts[i] - shifts(j, 0)));
            // Если элемент далеко от остальных сдвигов, запоминаем его
            if (min_dist > max_min_dist) {
                max_min_dist = min_dist;
                best_idx = i;
            }
        }
        //Лучший сдвиг добавляем
        if (best_idx >= 0) shifts(s++, 0) = all_shifts[best_idx];
        else shifts(s++, 0) = 0.5;
    }
    return shifts;
}
// Решение уравнения Риккати
Matrix solveRiccatiADI(const Matrix& A, const Matrix& B, const Matrix& Q,
                       const Matrix& R, const Matrix& E, int max_iter = 100,
                       double tau = 0.01, double tol = 1e-8) {
    const int n = A.getRows();
    Matrix P(n, n); // Начальное приближение
    double diff_norm;
      // Считаем сразу некоторые матрицы для упрощения
    Matrix E_trans = E.transpose();
    Matrix B_trans = B.transpose();
    Matrix R_inv = R.inverse();
    Matrix shifts = computeADIShifts(A, max_iter);

    for (int k = 0; k < max_iter; ++k) {
        Matrix P_prev = P;

        double alpha = shifts(k, 0);
        double beta = 1.0 - alpha;

        // Первый полушаг
        Matrix M1 = E + alpha * tau * A;
        Matrix M1_inv = M1.inverse();
        Matrix RHS1 = E_trans * P_prev * E - tau * Q + tau * (E_trans *
P_prev * B) * R_inv * (B_trans * P_prev * E);
        Matrix P_half = (M1_inv.transpose() * RHS1) * M1_inv;

        // Второй полушаг
        Matrix M2 = E + beta * tau * A;
        Matrix M2_inv = M2.inverse();
        Matrix RHS2 = E_trans * P_prev * E - tau * Q + tau * (E_trans *
P_half * B) * R_inv * (B_trans * P_prev * E);
        P = (M2_inv.transpose() * RHS2) * M2_inv;

        // Проверка сходимости
        diff_norm = 0;
        for (int i = 0; i < n; ++i)
            for (int j = 0; j < n; ++j)
                diff_norm += pow(P(i, j) - P_prev(i, j), 2);
        diff_norm = sqrt(diff_norm);

        if (diff_norm < tol) {
            cout<<"Convergence: "<<diff_norm<<endl;
            break;
        }
    }

    cout<<"Convergence: "<<diff_norm<<endl;
```

```
    return P;
}
int main() {
    try {
        Matrix A, B, C, E;
        string u="EABC_tests\\4x4"\\";
        A.readFromFile(u+"A.dat");
        B.readFromFile(u+"B.dat");
        C.readFromFile(u+"C.dat");
        E.readFromFile(u+"E.dat");

        int t = B.getCols();
        Matrix R(t, t);
        for (int i = 0; i < t; ++i) R(i, i) = 1;
        Matrix Q = C.transpose() * C;

        int n = A.getRows();
        double tau = 0.0001;
        Matrix P = solveRiccatiADI(A, B, Q, R, E, 100, tau);
        cout << "Solution P:\n" << P << endl;

    } catch (const exception& e) {
        cerr << "Error: " << e.what() << endl;
        return 1;
    }

    return 0;
}
```

ТЕСТИРОВАНИЕ

Проведено тестирование программы с помощью различных входных данных. Был использован компилятор C++ 3.4.2 и среда разработки Embarcadero Dev-C++ 6.3.

Шаг времени 0.0001 с, количество итераций 100.

Размер матрицы A	Вариант шагов	Сходимость	Время выполнения программы,с
8	1	0.000454674	0.2123
	2	0.000454676	0.4374
	3	0.000454675	0.6015
12	1	0.000567782	0.2605
	2	0.000567784	0.2404
	3	0.000567783	0.2293
20	1	0.0011778	0.4427
	2	0.0011778	0.3908
	3	0.0011778	0.4268
80	1	0.103294	11.09
	2	0.103271	10.96
	3	0.103278	10.89
120	1	0.172196	38.39
	2	0.172219	40.58
	3	0.172199	40.52
200	1	0.915907	189.8
	2	0.00140709	169.4
	3	0.00140709	172.9

Сравнивая три варианта упрощения шагов, можно сказать, что они примерно сравнимы по точности и времени выполнения, но с математической точки зрения коректнее выбирать 1 вариант.

Полные результаты тестирования есть в приложении А.

ЗАКЛЮЧЕНИЕ

Было изучено алгебраическое уравнение Риккати и численный метод ADI, написана блок-схема алгоритма программы, которая реализует решение уравнения Риккати, используя эвристический алгоритм подбора параметров и итерации Арнольда. На языке C++ написан код программы с выводом решения уравнения и его сходимости. Программа успешно протестирована на различных наборах данных.

Все задачи выполнены и цель достигнута.

ПРИЛОЖЕНИЕ А

Результаты тестирования

Тест 4 на 4

$$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E = \begin{pmatrix} 0.134 & 0.138 & 0.128 & 0.043 & -0.061 & -0.049 & -0.034 & 7.827 \times 10^{-8} \\ 0.138 & 0.145 & 0.136 & 0.049 & -0.049 & -0.057 & -0.034 & 8.104 \times 10^{-8} \\ 0.128 & 0.136 & 0.136 & 0.055 & -0.034 & -0.034 & -0.043 & 7.565 \times 10^{-8} \\ 0.043 & 0.049 & 0.055 & -4.673 \times 10^{-5} & -6.965 \times 10^{-5} & -7.939 \times 10^{-5} & -8.949 \times 10^{-5} & 0 \\ -0.061 & -0.049 & -0.034 & -6.966 \times 10^{-5} & 9.028 \times 10^{-3} & 7.125 \times 10^{-3} & 5.001 \times 10^{-3} & -4.845 \times 10^{-8} \\ -0.049 & -0.057 & -0.034 & -7.939 \times 10^{-5} & 7.125 \times 10^{-3} & 8.395 \times 10^{-3} & 4.957 \times 10^{-3} & -5.452 \times 10^{-8} \\ -0.034 & -0.034 & -0.043 & -8.948 \times 10^{-5} & 5.001 \times 10^{-3} & 4.957 \times 10^{-3} & 6.342 \times 10^{-3} & -5.976 \times 10^{-8} \\ 7.827 \times 10^{-8} & 8.104 \times 10^{-8} & 7.564 \times 10^{-8} & 0 & -4.845 \times 10^{-8} & -5.452 \times 10^{-8} & -5.976 \times 10^{-8} & 0 \end{pmatrix}$$

Рисунок А.1 – Результаты тестирования №1

Тест 6 на 6

$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E =$

	0	1	2	3	4	5
0	0.099	0.1	0.091	0.072	0.108	0.023
1	0.1	0.102	0.097	0.085	0.103	0.028
2	0.091	0.097	0.104	0.109	0.077	0.033
3	0.072	0.085	0.109	0.155	0.012	0.037
4	0.108	0.103	0.077	0.012	0.315	0.042
5	0.023	0.028	0.033	0.037	0.042	$-2.045 \cdot 10^{-5}$
6	-0.037	-0.022	$-6.008 \cdot 10^{-3}$	0.021	-0.057	$-4.989 \cdot 10^{-5}$
7	-0.022	-0.029	$-3.812 \cdot 10^{-3}$	0.022	-0.056	$-5.973 \cdot 10^{-5}$
8	$-6.264 \cdot 10^{-3}$	$-3.987 \cdot 10^{-3}$	-0.012	0.021	-0.051	$-6.96 \cdot 10^{-5}$
9	0.021	0.021	0.02	$6.246 \cdot 10^{-3}$	-0.042	$-7.982 \cdot 10^{-5}$
10	-0.057	-0.055	-0.05	-0.04	-0.089	$-9.016 \cdot 10^{-5}$
11	$3.797 \cdot 10^{-8}$	$3.699 \cdot 10^{-8}$	$3.027 \cdot 10^{-8}$	$1.089 \cdot 10^{-8}$	$1.129 \cdot 10^{-7}$	...

Рисунок А.2 – Результаты тестирования №2

Тест 10 на 10

$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E =$

	0	1	2	3	4	5
0	$9.625 \cdot 10^{-4}$	$-3.871 \cdot 10^{-4}$	$4.241 \cdot 10^{-3}$	$2.435 \cdot 10^{-3}$	$1.577 \cdot 10^{-3}$	$1.458 \cdot 10^{-4}$
1	$-3.873 \cdot 10^{-4}$	$7.677 \cdot 10^{-3}$	$4.609 \cdot 10^{-3}$	$5.131 \cdot 10^{-3}$	$3.388 \cdot 10^{-3}$	$3.654 \cdot 10^{-3}$
2	$4.241 \cdot 10^{-3}$	$4.609 \cdot 10^{-3}$	$9.39 \cdot 10^{-3}$	$7.518 \cdot 10^{-3}$	$6.509 \cdot 10^{-3}$	$3.761 \cdot 10^{-3}$
3	$2.435 \cdot 10^{-3}$	$5.131 \cdot 10^{-3}$	$7.518 \cdot 10^{-3}$	$6.035 \cdot 10^{-3}$	$3.928 \cdot 10^{-3}$	$2.875 \cdot 10^{-3}$
4	$1.577 \cdot 10^{-3}$	$3.388 \cdot 10^{-3}$	$6.509 \cdot 10^{-3}$	$3.928 \cdot 10^{-3}$	$4.017 \cdot 10^{-3}$	$2.344 \cdot 10^{-3}$
5	$1.456 \cdot 10^{-4}$	$3.654 \cdot 10^{-3}$	$3.761 \cdot 10^{-3}$	$2.875 \cdot 10^{-3}$	$2.344 \cdot 10^{-3}$	$-8.09 \cdot 10^{-4}$
6	$-1.264 \cdot 10^{-3}$	$6.391 \cdot 10^{-3}$	$3.636 \cdot 10^{-3}$	$2.985 \cdot 10^{-3}$	$3.787 \cdot 10^{-3}$	$1.682 \cdot 10^{-3}$
7	$4.311 \cdot 10^{-3}$	$-1.969 \cdot 10^{-4}$	$8.906 \cdot 10^{-3}$	$5.47 \cdot 10^{-3}$	$3.855 \cdot 10^{-3}$	$1.283 \cdot 10^{-3}$
8	$-1.011 \cdot 10^{-3}$	$2.22 \cdot 10^{-3}$	$4.012 \cdot 10^{-3}$	$2.674 \cdot 10^{-3}$	$5.047 \cdot 10^{-3}$	$6.633 \cdot 10^{-4}$
9	$1.13 \cdot 10^{-3}$	$9.974 \cdot 10^{-4}$	$1.752 \cdot 10^{-3}$	$1.146 \cdot 10^{-3}$	$1.312 \cdot 10^{-3}$	$1.459 \cdot 10^{-3}$
10	$-2.558 \cdot 10^{-3}$	$-1.333 \cdot 10^{-3}$	$1.985 \cdot 10^{-3}$	$4.797 \cdot 10^{-4}$	$-5.824 \cdot 10^{-4}$	$5.415 \cdot 10^{-4}$
11	$-1.543 \cdot 10^{-3}$	$1.325 \cdot 10^{-3}$	$1.538 \cdot 10^{-4}$	$1.391 \cdot 10^{-4}$	$-3.595 \cdot 10^{-4}$	...

Рисунок А.3 – Результаты тестирования №3

Тест 40 на 40

$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E =$

	0	1	2	3	4
0	$2.027 \cdot 10^{-4}$	$4.386 \cdot 10^{-5}$	$2.908 \cdot 10^{-3}$	$9.951 \cdot 10^{-4}$	$5.455 \cdot 10^{-3}$
1	$4.494 \cdot 10^{-5}$	$-4.696 \cdot 10^{-3}$	$-2.909 \cdot 10^{-3}$	$-1.908 \cdot 10^{-3}$	$7.464 \cdot 10^{-3}$
2	$2.909 \cdot 10^{-3}$	$-2.909 \cdot 10^{-3}$	$-1.219 \cdot 10^{-3}$	$-2.19 \cdot 10^{-3}$	$1.039 \cdot 10^{-3}$
3	$9.952 \cdot 10^{-4}$	$-1.909 \cdot 10^{-3}$	$-2.19 \cdot 10^{-3}$	$3.186 \cdot 10^{-4}$	$2.167 \cdot 10^{-3}$
4	$5.454 \cdot 10^{-3}$	$7.465 \cdot 10^{-3}$	$1.039 \cdot 10^{-3}$	$2.167 \cdot 10^{-3}$	$-9.341 \cdot 10^{-3}$
5	$-1.147 \cdot 10^{-3}$	$5.254 \cdot 10^{-3}$	$6.593 \cdot 10^{-3}$	$3.72 \cdot 10^{-3}$	$6.143 \cdot 10^{-3}$
6	$-6.82 \cdot 10^{-4}$	$1.638 \cdot 10^{-3}$	$3.202 \cdot 10^{-3}$	$1.661 \cdot 10^{-3}$	$4.102 \cdot 10^{-3}$
7	$1.815 \cdot 10^{-3}$	0.011	$8.045 \cdot 10^{-3}$	$4.568 \cdot 10^{-3}$	$-1.282 \cdot 10^{-3}$
8	$1.928 \cdot 10^{-3}$	$-5.74 \cdot 10^{-3}$	$-4.81 \cdot 10^{-3}$	$-2.206 \cdot 10^{-3}$	$2.822 \cdot 10^{-3}$
9	$1.688 \cdot 10^{-3}$	0.01	$5.926 \cdot 10^{-3}$	$5.632 \cdot 10^{-3}$	$-3.791 \cdot 10^{-3}$
10	$1.258 \cdot 10^{-3}$	$1.201 \cdot 10^{-3}$	$2.54 \cdot 10^{-4}$	$8.363 \cdot 10^{-4}$	$2.788 \cdot 10^{-4}$
11	$3.313 \cdot 10^{-3}$	$9.268 \cdot 10^{-3}$	$3.939 \cdot 10^{-3}$	$3.358 \cdot 10^{-3}$	$-5.373 \cdot 10^{-3}$
12	$-8.877 \cdot 10^{-4}$	$-4.64 \cdot 10^{-4}$	$2.837 \cdot 10^{-3}$	$2.158 \cdot 10^{-3}$	$6.682 \cdot 10^{-3}$
13	$3.09 \cdot 10^{-3}$	$1.766 \cdot 10^{-4}$	$-2.509 \cdot 10^{-3}$	$-9.407 \cdot 10^{-4}$	$-2.652 \cdot 10^{-3}$
14	$-2.014 \cdot 10^{-3}$	$-1.05 \cdot 10^{-4}$	$2.702 \cdot 10^{-3}$	$1.271 \cdot 10^{-3}$	$8.182 \cdot 10^{-3}$
15	$5.682 \cdot 10^{-4}$	$-2.114 \cdot 10^{-3}$	$-1.89 \cdot 10^{-3}$	$-5.643 \cdot 10^{-4}$	...

Рисунок А.4 – Результаты тестирования №4

Тест 60 на 60

$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E =$

	0	1	2	3	4
0	0.423	0.404	0.378	0.415	0.389
1	0.404	0.521	0.411	0.484	0.411
2	0.378	0.411	0.407	0.422	0.394
3	0.415	0.484	0.422	0.526	0.433
4	0.389	0.411	0.394	0.433	0.436
5	0.339	0.342	0.317	0.344	0.334
6	0.236	0.255	0.23	0.264	0.245
7	0.321	0.337	0.28	0.335	0.288
8	0.377	0.438	0.392	0.46	0.404
9	0.29	0.307	0.271	0.315	0.273
10	0.338	0.377	0.348	0.386	0.363
11	0.414	0.461	0.423	0.461	0.443
12	0.429	0.473	0.421	0.469	0.416
13	0.361	0.387	0.343	0.391	0.349
14	0.386	0.436	0.39	0.441	0.412
15	0.381	0.402	0.377	0.427	...

Рисунок А.5 – Результаты тестирования №5

Тест 100 на 100

$E^T \cdot P \cdot A + A^T \cdot P \cdot E - E^T \cdot P \cdot B \cdot B^T \cdot P \cdot E =$

	0	1	2	3	4
0	$-6.375 \cdot 10^{-10}$	0	0	0	0
1	0	$-6.375 \cdot 10^{-10}$	0	0	0
2	0	0	$-6.375 \cdot 10^{-10}$	0	0
3	0	0	0	$-6.375 \cdot 10^{-10}$	0
4	0	0	0	0	$-6.375 \cdot 10^{-10}$
5	0	0	0	0	0
6	0	0	0	0	0
7	0	0	0	0	0
8	0	0	0	0	0
9	0	0	0	0	0
10	0	0	0	0	0
11	0	0	0	0	0
12	0	0	0	0	0
13	0	0	0	0	0
14	0	0	0	0	0
15	0	0	0	0	...

Рисунок А.6 – Результаты тестирования №6