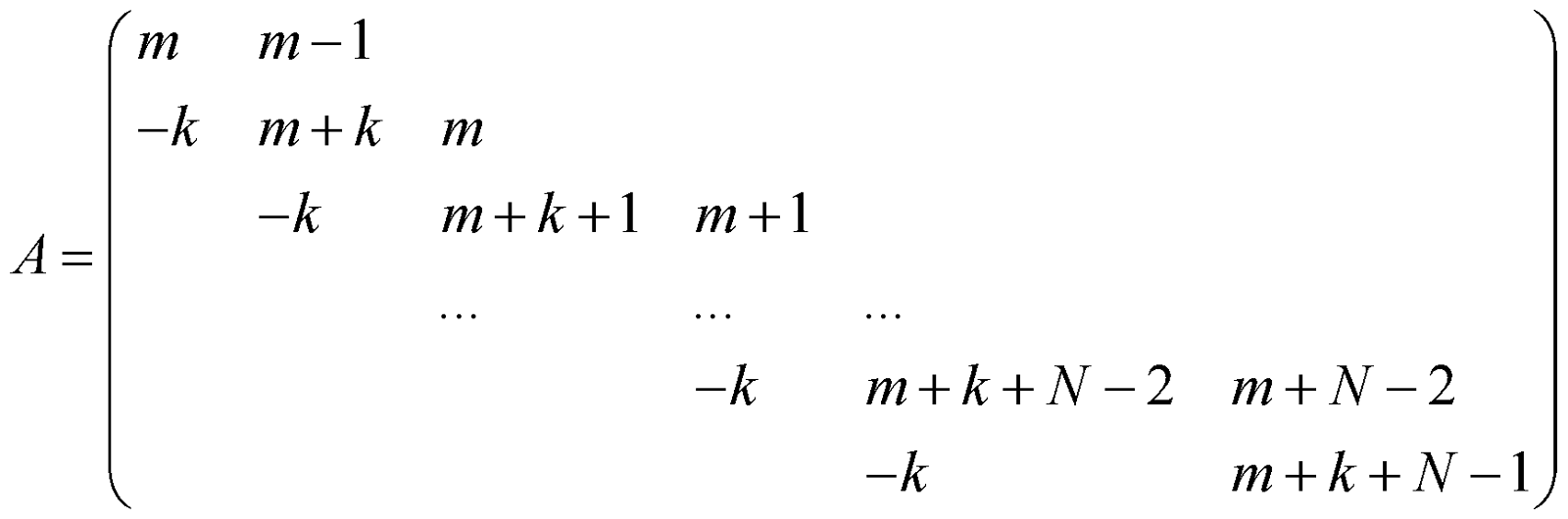
**Лабораторная работа 3 «Решение СЛАУ методом прогонки»**

Выполнил студент 3 курса 4 группы ФПМИ БГУ Видевич Александр

**Задача.**

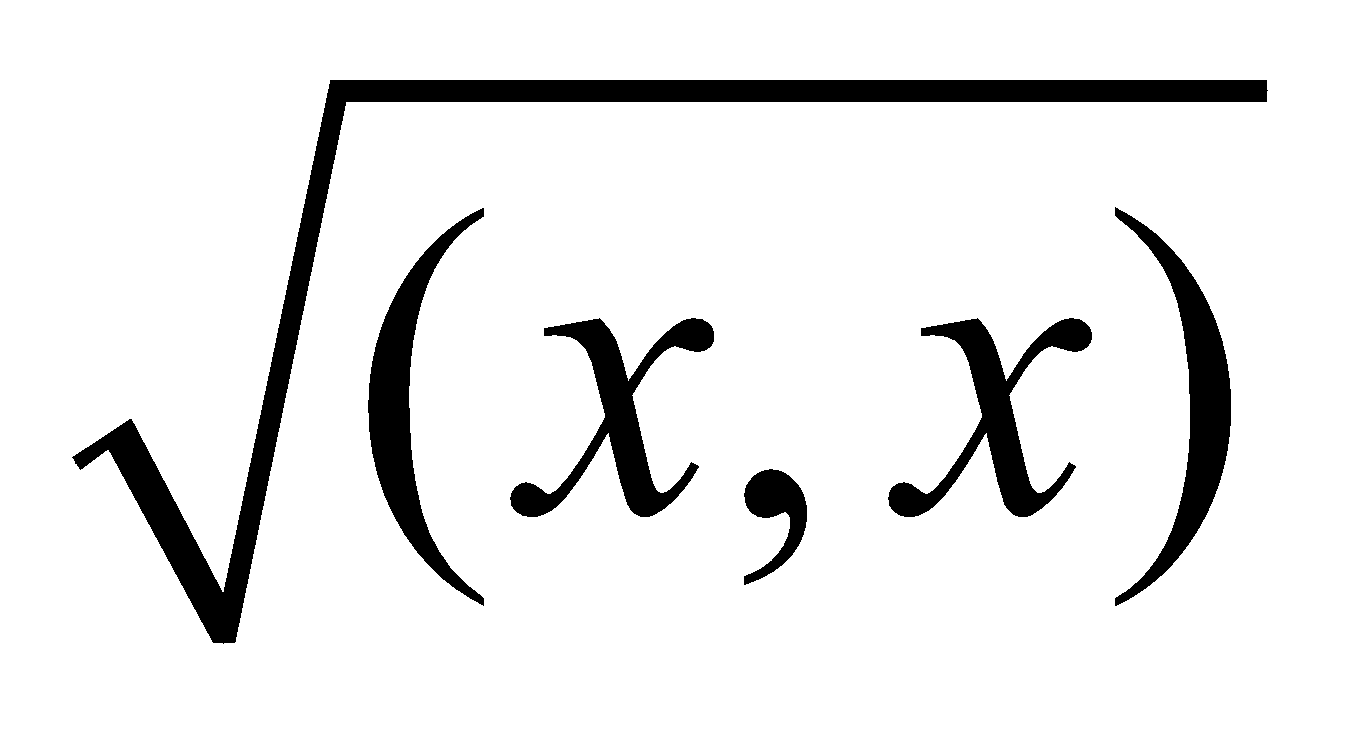
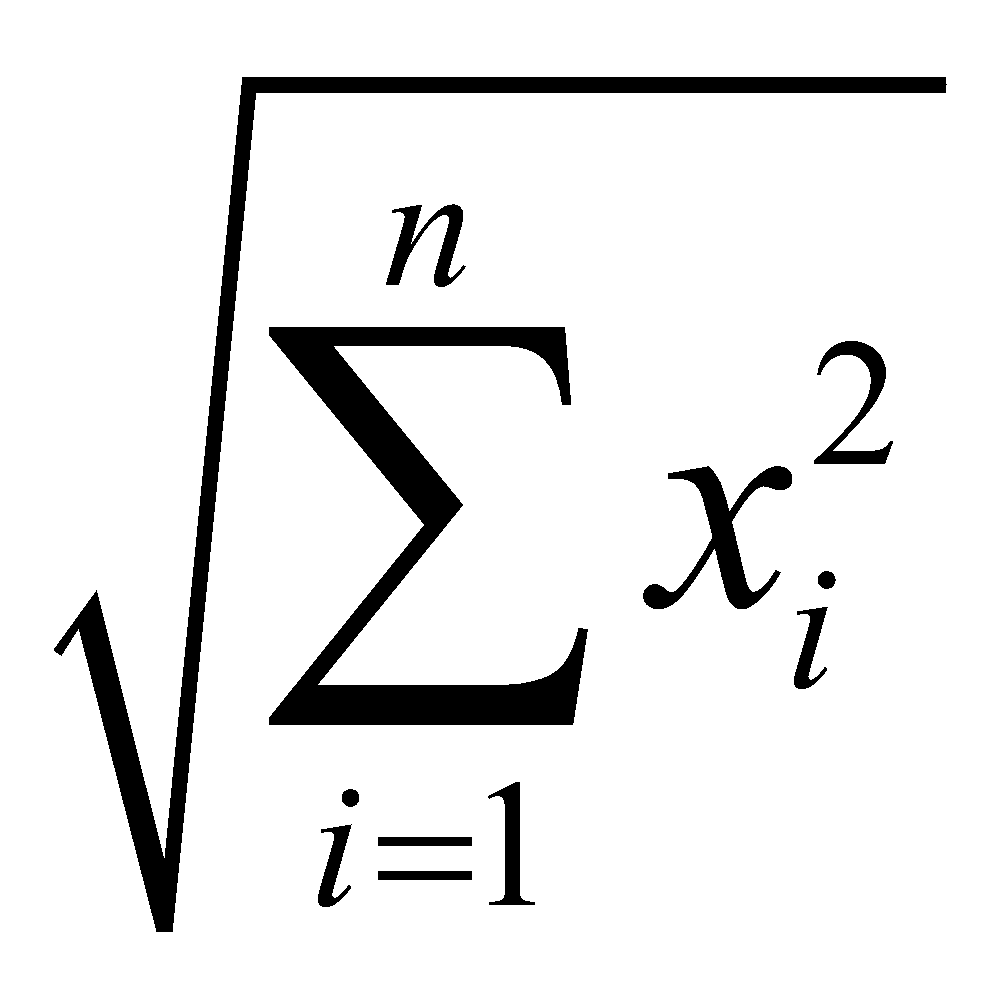
Разработать программу численного решения методом прогонки СЛАУ *Ay=f* вида. Матрицу системы задать следующим образом:

*.*

Правую часть *f* задать умножением матрицы *A* на вектор *y =* (1, 2, ..., *N*+1): *f=Ay*.

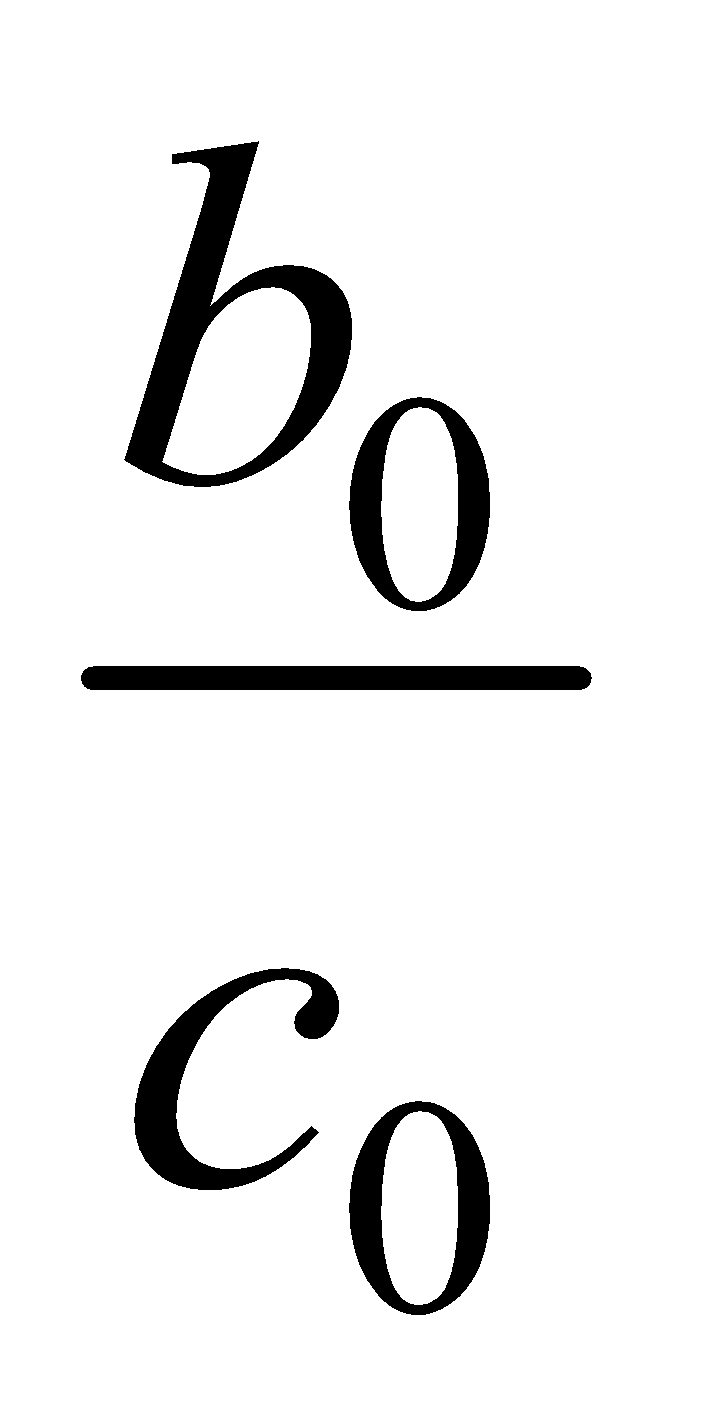
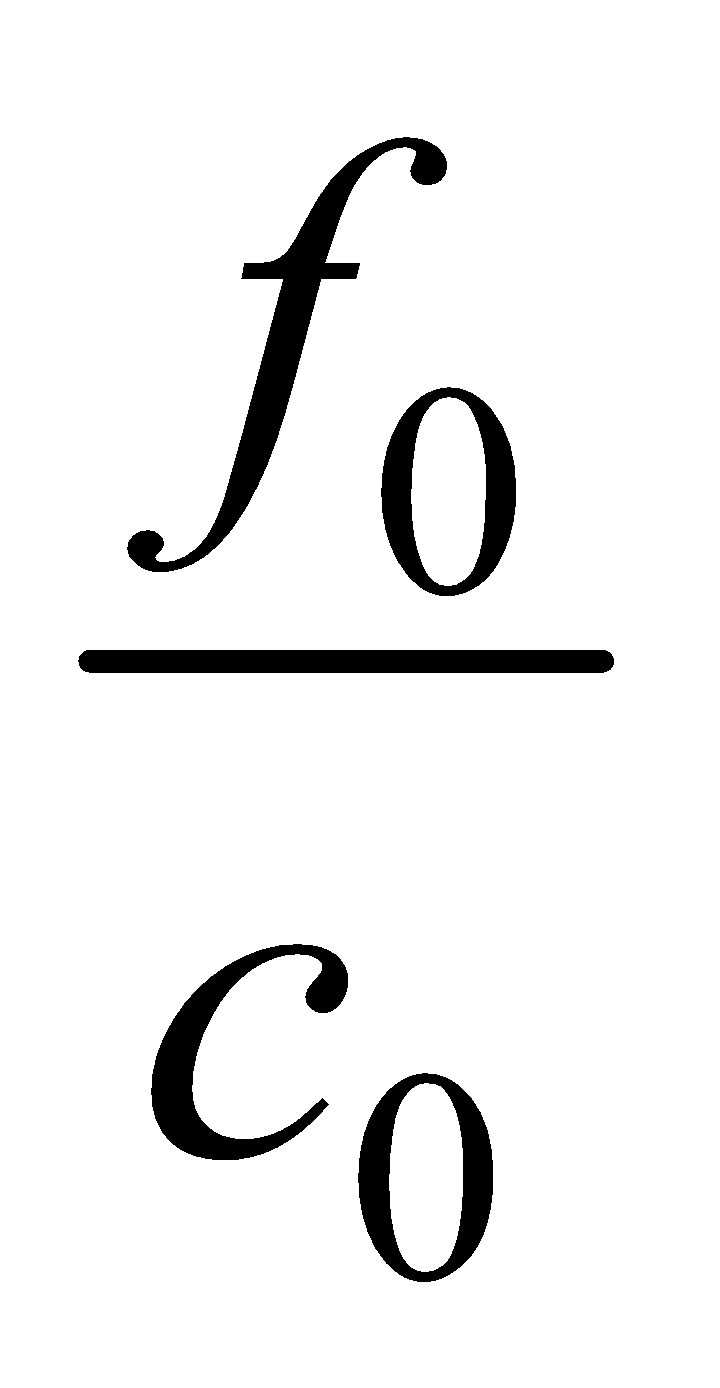
Для вычислений выбрать параметры:

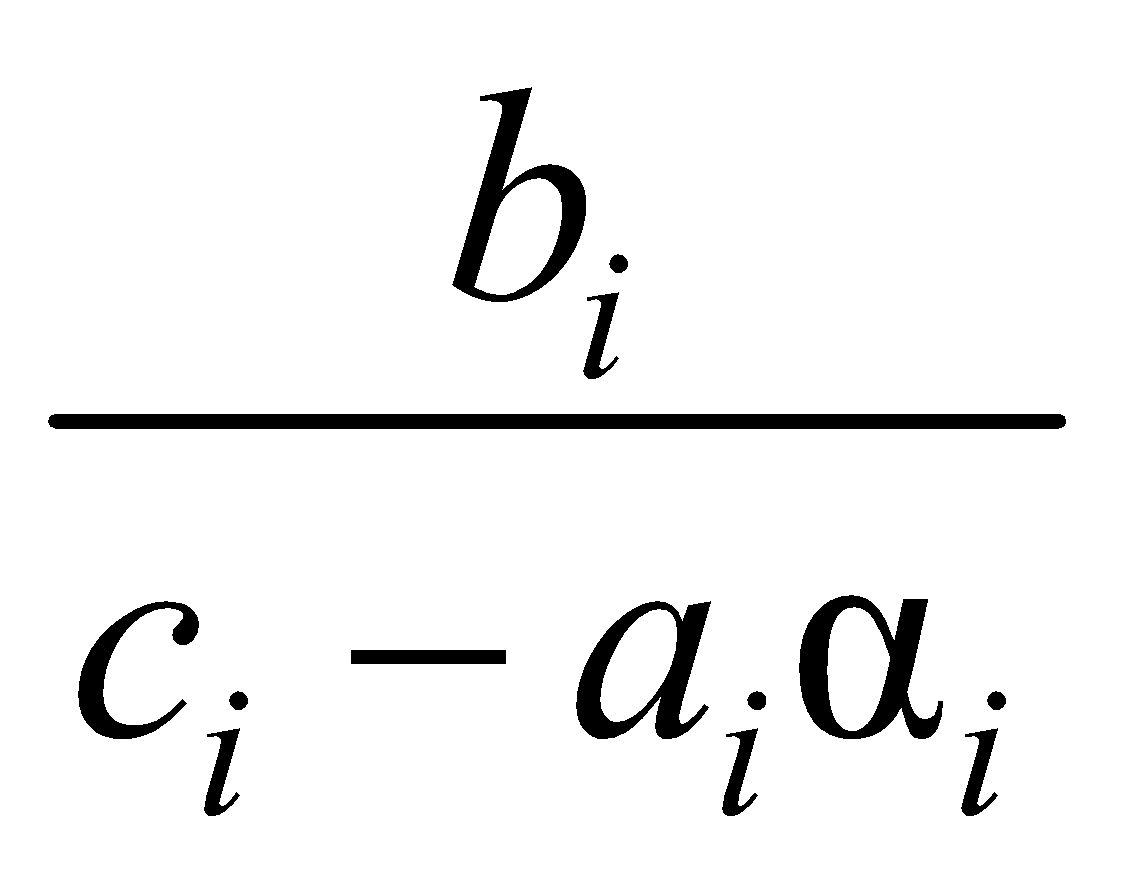
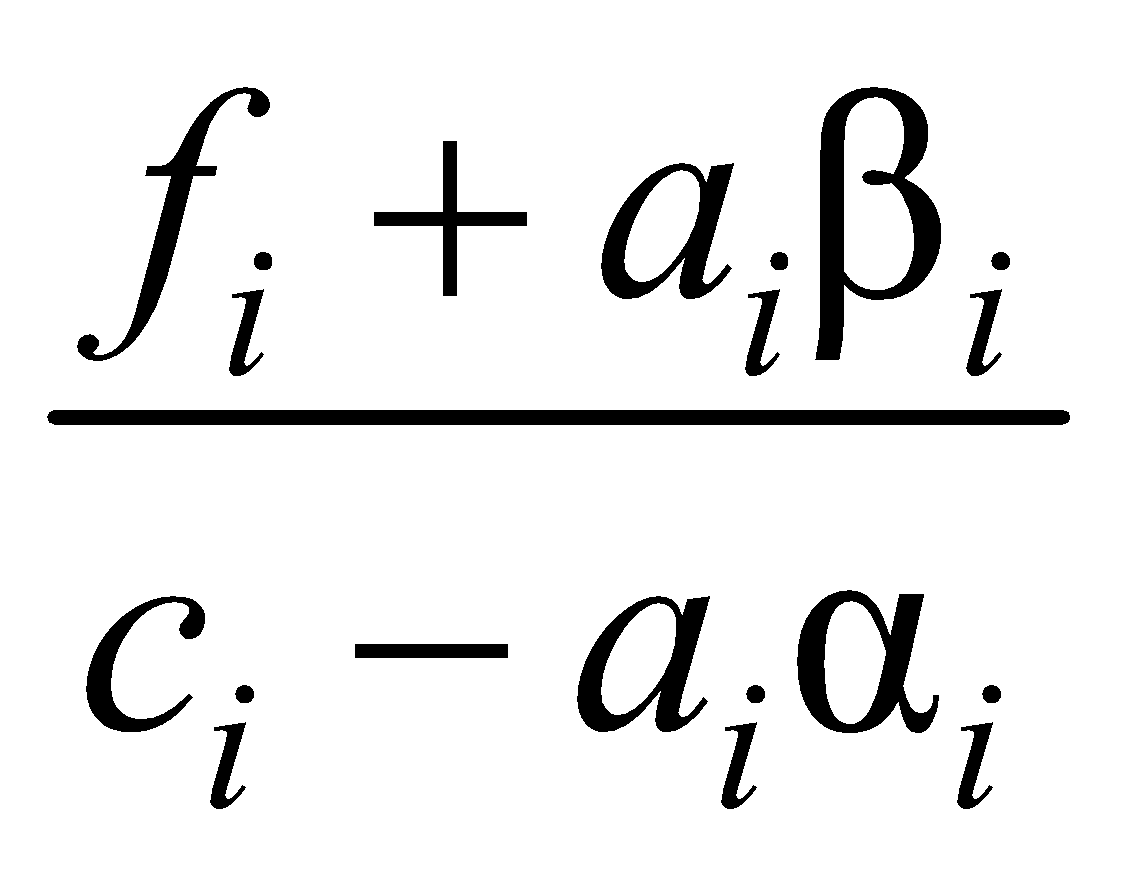
* *N*+1 – (порядок матрицы) одно из чисел в пределах от 1500 до 2000;
* *m* – номер в списке студенческой группы;
* *k* – номер студенческой группы.

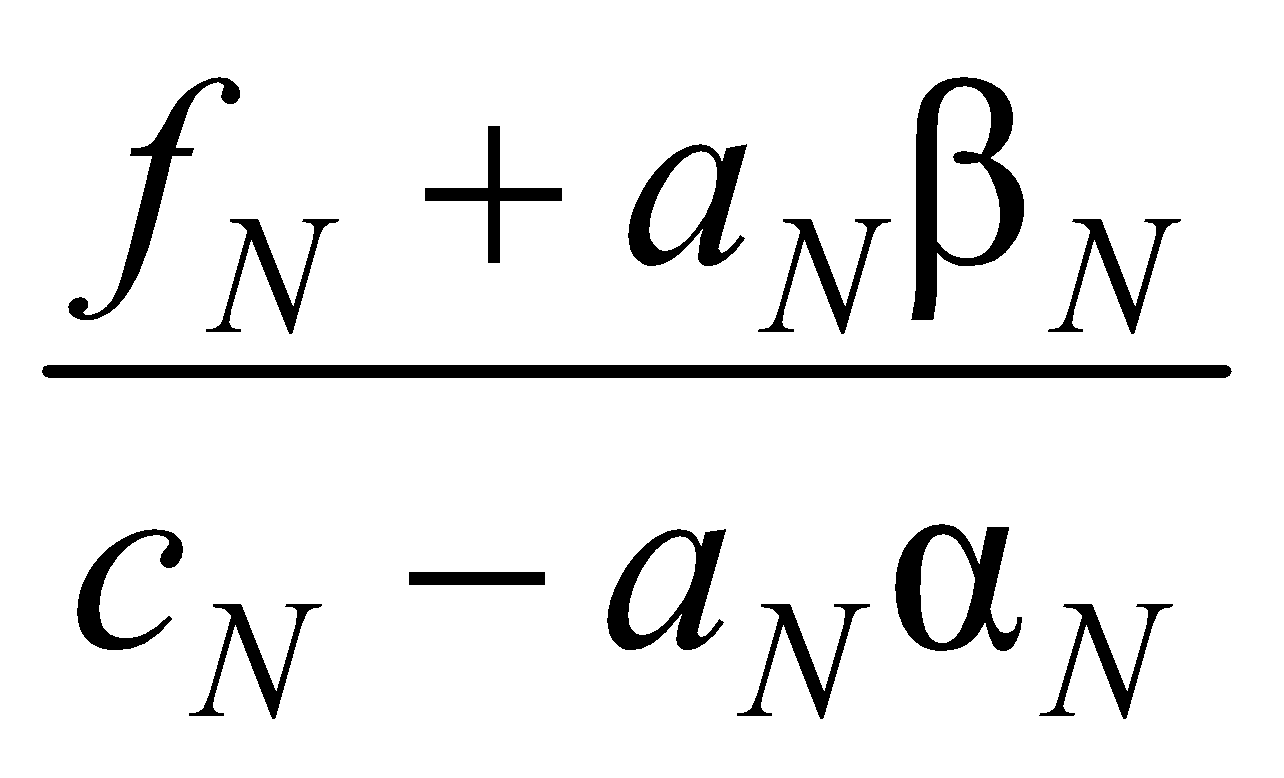
Для оценки погрешности вычислений использовалась евклидова (сферическая норма): ||*x*||2==.

Относительная погрешность , где – точное решение (*m*, *m*+1, ..., *n*+*m*–1).

Прямая прогонка – вычисление прогоночных коэффициентов по формулам:

α1*=**,* β1*=**,*

α*i+*1*=**,* β*i+*1*=**, i=* 1, 2, ..., *N*–1

β*N+*1*=**;*

Обратная прогонка – вычисление решения по формулам:

*yN*=β*N+*1, *yi=*α*i+*1*yi+*1*+*β*i+*1, *i=* *N*–1, ...,1, 0.

**Входные данные.**

N = 1800, m = 4, k = 4

**Листинг программы.**

*#include* <iostream>

*#include* <vector>

*#include* <ctime>

*#include* <chrono>

*#include* <cmath>

int n;

*// функция создания трехдиагональной матрицы*

*#define* M *4*

*#define* K *4*

void GenerateMatrix(std::vector<float> *&*b, std::vector<float> *&*c, std::vector<float> *&*a)

{

    c[*0*] = M;

    b[*0*] = M - *1*;

*for* (int i = *1*; i < n; ++i)

    {

        a[i - *1*] = -K;

        b[i] = b[i - *1*] + *1*;

        c[i] = b[i] + K;

    }

    a[n - *1*] = -K;

    c[n] = M + K + n - *1*;

}

std::vector<std::vector<float>> CreateMatrixForGauss(*const* std::vector<float> *&*b, *const* std::vector<float> *&*c, *const* std::vector<float> *&*a)

{

    std::vector<std::vector<float>> matrixForGauss(n + *1*, std::vector<float>(n + *1*, *0*));

    matrixForGauss[*0*][*0*] = c[*0*];

    matrixForGauss[*0*][*1*] = b[*0*];

*for* (int i = *1*; i < n; ++i)

    {

        matrixForGauss[i][i - *1*] = a[i];

        matrixForGauss[i][i] = c[i];

        matrixForGauss[i][i + *1*] = b[i];

    }

    matrixForGauss[n][n] = c[n];

    matrixForGauss[n][n - *1*] = a[n - *1*];

*return* matrixForGauss;

}

*// функция создания вектора y*

std::vector<int> GenerateVector()

{

    std::vector<int> vector(n + *1*);

*for* (int i = *0*; i < n + *1*; ++i)

    {

        vector[i] = i + *1*;

    }

*return* vector;

}

*// функция умножения трехдиагональной матрицы на вектор*

template <typename T>

std::vector<float> MatrixVectorMultiply(*const* std::vector<float> *&*b, *const* std::vector<float> *&*c, *const* std::vector<float> *&*a, *const* std::vector<T> *&*vector)

{

    std::vector<float> result(n + *1*);

    result[*0*] = c[*0*] \* vector[*0*] + b[*0*] \* vector[*1*];

*for* (int i = *1*; i < n; i++)

    {

        result[i] = a[i - *1*] \* vector[i - *1*] + c[i] \* vector[i] + b[i] \* vector[i + *1*];

    }

    result[n] = a[n - *1*] \* vector[n - *1*] + c[n] \* vector[n];

*return* result;

}

*// функция осществления прямой прогоки*

void ForwardRunThrough(std::vector<float> *&*b, std::vector<float> *&*c, std::vector<float> *&*a, std::vector<float> *&*vector)

{

    vector[*0*] /= c[*0*];

    b[*0*] /= -c[*0*];

*for* (int i = *1*; i < n; ++i)

    {

        b[i] /= -(c[i] + a[i - *1*] \* b[i - *1*]);

        vector[i] = (vector[i] - a[i - *1*] \* vector[i - *1*]) / (c[i] + a[i - *1*] \* b[i - *1*]);

    }

    vector[n] = (vector[n] - a[n - *1*] \* vector[n - *1*]) / (c[n] + a[n - *1*] \* b[n - *1*]);

}

*// функция осществления обратной прогоки*

std::vector<float> ReverseRunThrough(*const* std::vector<float> *&*b, *const* std::vector<float> *&*vector)

{

    std::vector<float> solution(n + *1*);

    solution[n] = vector[n];

*for* (int i = n - *1*; i >= *0*; --i)

    {

        solution[i] = b[i] \* solution[i + *1*] + vector[i];

    }

*return* solution;

}

*// функция решения СЛАУ методом прогонки*

std::vector<float> SolveSystemUsingRunThroughMethod(std::vector<float> b, std::vector<float> c, std::vector<float> a, std::vector<float> vector)

{

    ForwardRunThrough(b, c, a, vector);

*return* ReverseRunThrough(b, vector);

}

*// функция шага прямого хода метода Гауса*

void MakeMove(std::vector<std::vector<float>> *&*matrix, std::vector<float> *&*vector, *const* int k)

{

*// обход всех нижестоящих уравнений системы*

*for* (int i = k + *1*; i < n; i++)

    {

*// получение коэффициента lik*

        float lik = matrix[i][k] / matrix[k][k];

        matrix[i][k] = *0.0*;

*// получение новой системы*

*for* (int j = k + *1*; j < n; j++)

        {

            matrix[i][j] -= lik \* matrix[k][j];

        }

        vector[i] -= lik \* vector[k];

    }

}

*// функция осуществления обратного хода метода Гауса*

std::vector<float> GetGaussResult(*const* std::vector<std::vector<float>> *&*matrix, *const* std::vector<float> *&*vector)

{

*// создание вектора (x) для хранения результата*

    std::vector<float> solution(n);

*// получение элемента xn*

    solution[n - *1*] = vector[n - *1*] / matrix[n - *1*][n - *1*];

*for* (int i = n - *2*; i >= *0*; --i)

    {

*// получение элемента xi*

        float sum = *0.0*;

*for* (int j = i + *1*; j < n; ++j)

        {

            sum += matrix[i][j] \* solution[j];

        }

        solution[i] = (vector[i] - sum) / matrix[i][i];

    }

*// возвращение результата*

*return* solution;

}

std::vector<float> RunGaussWithoutSelectingLeadingElement(*const* std::vector<std::vector<float>> *&*matrix, *const* std::vector<float> *&*vector)

{

*// создание копий данных для предотвращения их изменения*

    std::vector<std::vector<float>> matrixCopy(matrix);

    std::vector<float> vectorCopy(vector);

*// проход по каждому уравнению системы*

*for* (int k = *0*; k < n - *1*; ++k)

    {

*// прямой ход метода Гауса для текущего шага*

        MakeMove(matrixCopy, vectorCopy, k);

    }

*// обратный ход метода Гауса и возвращение результата*

*return* GetGaussResult(matrixCopy, vectorCopy);

}

*// функция вывода вектора*

template <typename T>

void PrintVector(*const* std::vector<T> *&*vector, *const* int numberOfElements = *0*, *const* std::string *&*message = "")

{

*if* (message != "")

    {

        std::cout << message << ' ';

    }

*const* int border = numberOfElements != *0* ? numberOfElements : vector.size();

*for* (int i = *0*; i < border; ++i)

    {

        std::cout << vector[i] << ' ';

    }

    std::cout << '*\n*';

}

template <typename T>

float CalculateEuclideanNorm(*const* std::vector<T> *&*vector)

{

    float sumOfSquares = *0.0*;

*// подсчет суммы квадратов координат вектора*

*for* (auto element : vector)

    {

        sumOfSquares += static\_cast<float>(element \* element);

    }

*// извелечение корня из суммы квадратов и возвращение результата*

*return* std::sqrt(sumOfSquares);

}

*// функция подсчета относительной погрешности*

float GetRelativeError(*const* std::vector<int> *&*originalVector, *const* std::vector<float> *&*calculatedVector)

{

    std::vector<float> temp(n);

*// подсчет вектора x-x\**

*for* (int i = *0*; i < n; ++i)

    {

        temp[i] = originalVector[i] - calculatedVector[i];

    }

*// подсчет норм и возвращение результата*

*return* CalculateEuclideanNorm(temp) / CalculateEuclideanNorm(originalVector);

}

int main()

{

    std::ios\_base::sync\_with\_stdio(*false*);

    std::cin.tie(*nullptr*);

    std::srand(static\_cast<unsigned int>(std::time(*nullptr*)));

*// получение данных о размерности задачи*

    std::cin >> n;

*// генерация данных*

    std::vector<float> b(n);

    std::vector<float> c(n + *1*);

    std::vector<float> a(n);

    GenerateMatrix(b, c, a);

    std::vector<std::vector<float>> matrixForGauss = CreateMatrixForGauss(b, c, a);

    std::vector<int> y = GenerateVector();

    std::vector<float> f = MatrixVectorMultiply(b, c, a, y);

*// решение СЛАУ методом прогонки и подсчет времени выполнения*

    auto start = std::chrono::high\_resolution\_clock::now();

    std::vector<float> s = SolveSystemUsingRunThroughMethod(b, c, a, f);

    auto end = std::chrono::high\_resolution\_clock::now();

*const* int time1 = std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count();

    start = std::chrono::high\_resolution\_clock::now();

    RunGaussWithoutSelectingLeadingElement(matrixForGauss, f);

    end = std::chrono::high\_resolution\_clock::now();

*const* int time2 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count();

*// Вывод вектора приближенного решения*

    PrintVector(s, *5*, "Vector y\*:");

*// Вывод относительной погрешности*

    std::cout << GetRelativeError(y, s) << '*\n*';

*// Вывод времени выполнения*

    std::cout << "Time (RunThroughMethod): " << time1 << "mcs*\n*";

    std::cout << "Time (Gauss): " << time2 << "ms*\n*";

*return* *0*;

}

**Выходные данные.**

Vector y\*: 1 2 3 4 5

Relative error: 4.11373e-07

Time (RunThroughMethod): 48mcs

Time (Gauss): 11719ms

**Выводы.**

Применение метода прогонки для работы с трехдиагональной матрицами дает значительный выигрыш во времени.