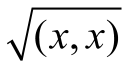
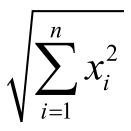
**Лабораторная работа №1 Метод Гауса**Выполнил студент 3 курса 4 группы ФПМИ БГУ Видевич Александр

**Задача.**

Разработать программу численного решения СЛАУ методом Гаусса без выбора ведущего элемента и с выбором ведущего элемента по столбцу. Исходная матрица должна содержать случайные числа из диапазона от -1000 до 1000. Правую часть b задается умножением матрицы A (размера n\*n) на вектор x = (m, m+1, ..., n+m–1): b=A\*x, где m - номер в списке группы. Для оценки погрешности вычислений использовалась евклидова (сферическая норма): ||*x*||2==.

Норма вектора невязки 

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

**Входные данные.**

n = 1800, m = 4

**Листинг программы.**

*#include* <iostream>

*#include* <vector>

*#include* <stdlib.h>

*#include* <ctime>

*#include* <cmath>

*#include* <chrono>

*#include* <string>

int n;

*// функция генерации матрицы*

std::*vector*<std::*vector*<float>> GenerateMatrix(*const* int size)

{

    std::*vector*<std::*vector*<float>> matrix(size, std::*vector*<float>(size));

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

    {

*for* (int j = *0*; j < size; ++j)

        {

            matrix[i][j] = -*1000.0f* + static\_cast<float>(std::rand()) /

                                          (static\_cast<float>(RAND\_MAX /

                                                              (*2000.0f*)));

        }

    }

*return* matrix;

}

*// фунция создания вектора x*

*#define* M *4*

std::*vector*<int> GetVector(*const* int size)

{

    std::*vector*<int> vector(size);

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

    {

        vector[i] = M + i;

    }

*return* vector;

}

*// функция умножения матрицы на вектор*

template <typename T>

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

{

*const* int size = vector.size();

    std::*vector*<float> result(size);

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

    {

        float sum = *0*;

*for* (int j = *0*; j < size; j++)

        {

            sum += matrix[i][j] \* static\_cast<float>(vector[j]);

        }

        result[i] = sum;

    }

*return* result;

}

*// 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> GaussWithoutSelectingLeadingElement(*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 нахождения максимального по модулю элемента для шага k*

int FindRowWithMaxElement(*const* std::*vector*<std::*vector*<float>> *&*matrix, *const* int startRow)

{

*// задание начальных значений*

*const* int k = startRow;

    int maxRowIndex = startRow;

    float maxElement = std::abs(matrix[k][k]);

*// обход нижестоящих уравнений и нахождение в них максимального элемента*

*for* (int i = k + *1*; i < matrix.size(); i++)

    {

        float absElement = std::abs(matrix[i][k]);

*if* (absElement > maxElement)

        {

            maxRowIndex = i;

            maxElement = absElement;

        }

    }

*// возвращение индекса строки с максимальным по модулю элементом*

*return* maxRowIndex;

}

*// функция обмена строк*

void SwapRows(std::*vector*<std::*vector*<float>> *&*matrix, std::*vector*<float> *&*vector, *const* int i, *const* int j)

{

*// смена строк матрицы A*

    std::swap(matrix[i], matrix[j]);

*// смена координат вектора b*

    std::swap(vector[i], vector[j]);

}

*// фукция решения СЛАУ методом Гауса с выбором ведущего элемента*

std::*vector*<float> GaussWithSelectingLeadingElement(*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++)

    {

*// получение индекса строки с максимальным по модулю элементом в рамках текущего шага*

*const* int max = FindRowWithMaxElement(matrixCopy, k);

*// перестановка строк*

        SwapRows(matrixCopy, vectorCopy, k, max);

*// 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*';

}

*// функция подсчета квадратичной (евклидовой нормы)*

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 нормы вектора невязки*

float GetNormOfResidualVector(*const* std::*vector*<std::*vector*<float>> *&*matrix, *const* std::*vector*<float> *&*calculatedVector, *const* std::*vector*<float> *&*f)

{

    std::*vector*<float> ax = MatrixVectorMultiply(matrix, calculatedVector);

*// подсчет вектора f-Ax\**

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

    {

        ax[i] = f[i] - ax[i];

    }

*// подсчет нормы вектора и возвращение результата*

*return* CalculateEuclideanNorm(ax);

}

*// 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*)));

*// ввод данных*

    std::cin >> n;

*// генерация задачи*

*const* std::*vector*<std::*vector*<float>> matrix = GenerateMatrix(n);

*const* std::*vector*<int> vector = GetVector(n);

*const* std::*vector*<float> b = MatrixVectorMultiply(matrix, vector);

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

*// использование метода Гауса без выбора главного элемента*

*const* std::*vector*<float> res1 = GaussWithoutSelectingLeadingElement(matrix, b);

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

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

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

*// использование метода Гауса с выбором главного элемента*

*const* std::*vector*<float> res2 = GaussWithSelectingLeadingElement(matrix, b);

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

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

*// вывод результатов*

    PrintVector(vector, *5*, "Original vector x:");

    PrintVector(res1, *5*, "Vector x calculated without selecting leading element:");

    PrintVector(res2, *5*, "Vector x calculated with selecting leading element:");

*// подсчет и вывод нормы ветора невязки*

    std::cout << "Norm of residual vector (result calculated without selecting leading element): " << GetNormOfResidualVector(matrix, res1, b) << '*\n*';

    std::cout << "Norm of residual vector (result calculated with selecting leading element): " << GetNormOfResidualVector(matrix, res2, b) << '*\n*';

*// подсчет и вывод относительной погрешности*

    std::cout << "RelativeError (result calculated without selecting leading element): " << GetRelativeError(vector, res1) << '*\n*';

    std::cout << "RelativeError (result calculated with selecting leading element): " << GetRelativeError(vector, res2) << '*\n*';

*// вывод времени выполнения*

    std::cout << "Time (result calculated without selecting leading element): " << time1 << "ms*\n*";

    std::cout << "Time (result calculated with selecting leading element): " << time2 << "ms*\n*";

*return* *0*;

}

**Выходные данные.**

Original vector x: 4 5 6 7 8

Vector x calculated without selecting leading element: 101.301 -173.892 623.776 73.9223 238.203

Vector x calculated with selecting leading element: 3.92072 9.51459 13.5079 4.34584 9.22229

Norm of residual vector (result calculated without selecting leading element): 4.22546e+07

Norm of residual vector (result calculated with selecting leading element): 48216.3

RelativeError (result calculated without selecting leading element): 0.363054

RelativeError (result calculated with selecting leading element): 0.0066961

Time (result calculated without selecting leading element): 29404ms

Time (result calculated with selecting leading element): 32436ms

**Выводы.**

Метод Гаусса является неустойчивым по входным данным: в случае относительной малости главного элемента процесс вычислений приводит к сильному накоплению погрешностей. В то же время метод Гаусса с выбором ведущего элемента является устойчивым.