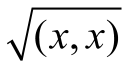
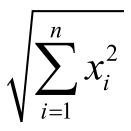
**Задача.**

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 = 1700, m = 14

**Листинг программы.**

#include <iostream>  
#include <vector>  
#include <stdlib.h>  
#include <chrono>  
  
std::vector<double> generateVectorB(size\_t size) {  
 std::vector<double> ans(size**,** 0)**;** for (size\_t i = 0**;** i < size**;** ++i) {  
 ans[i] = 14 + i**;** }  
 return ans**;**}  
  
std::vector<double> multiply(std::vector<std::vector<double>> A**,** std::vector<double> x) {  
 std::vector<double> ans(x.size()**,** 0)**;** {  
 for (int i = 0**;** i < x.size()**;** i++) {  
 double sum = 0**;** for (int j = 0**;** j < x.size()**;** j++) {  
 sum += A[i][j] \* x[j]**;** }  
 ans[i] = sum**;** }  
 }  
 return ans**;**}  
  
std::vector<std::vector<double>> generateMatrix(int size) {  
 std::vector<double> tmp(size**,** 0)**;** std::vector<std::vector<double>> result(size**,** tmp)**;** for (int i = 0**;** i < size**;** i++) {  
 for (int j = 0**;** j < size**;** j++)  
 result[i][j] = rand() % 2000 - 1000**;** }  
 return result**;**}  
  
std::vector<double> gaussWithoutMainElem(std::vector<std::vector<double>> matrix**,** std::vector<double> b) {  
 std::vector<std::vector<double>> matrix\_copy(b.size()**,** std::vector<double>(b.size()**,** 0))**;** std::vector<double> b\_copy(b.size()**,** 0)**;** for(int i = 0**;** i < b.size()**;** ++i) {  
 for(int j = 0**;** j < b.size()**;** ++j) {  
 matrix\_copy[i][j] = matrix[i][j]**;** }  
 b\_copy[i] = b[i]**;** }  
  
 for(int k = 0**;** k < b.size() - 1**;** ++k) {  
 for(int i = k + 1**;** i < b.size()**;** ++i) {  
 double l = matrix\_copy[i][k] / matrix\_copy[k][k]**;** matrix\_copy[i][k] = 0.0f**;** for(int j = k + 1**;** j < b.size()**;** ++j) {  
 matrix\_copy[i][j] -= l \* matrix\_copy[k][j]**;** }  
 b\_copy[i] -= l \* b\_copy[k]**;** }  
 }  
  
 std::vector<double> x(b.size()**,** 0)**;** x[b.size() - 1] = b\_copy[b.size() - 1] / matrix\_copy[b.size() - 1][b.size() - 1]**;** for(int i = b.size() - 2**;** i >= 0**;** --i) {  
 double sum = 0.0f**;** for(int j = i + 1**;** j < b.size()**;** ++j) {  
 sum += matrix\_copy[i][j] \* x[j]**;** }  
 x[i] = (b\_copy[i] - sum) / matrix\_copy[i][i]**;** }  
  
 return x**;**}  
  
std::vector<double> gaussWithMainElem(std::vector<std::vector<double>> matrix**,** std::vector<double> b) {  
 //Создание копий  
 std::vector<std::vector<double>> matrix\_copy(b.size()**,** std::vector<double>(b.size()**,** 0))**;** std::vector<double> b\_copy(b.size()**,** 0)**;** for(int i = 0**;** i < b.size()**;** ++i) {  
 for(int j = 0**;** j < b.size()**;** ++j) {  
 matrix\_copy[i][j] = matrix[i][j]**;** }  
 b\_copy[i] = b[i]**;** } //Прямой ход  
 for(int k = 0**;** k < b.size() - 1**;** ++k) {  
 double max = std::abs(matrix\_copy[k][k])**;** int coord\_str = 0**;** for(int i = k + 1**;** i < b.size()**;** ++i) {  
 if(std::abs(matrix\_copy[i][k]) > max) {  
 max = std::abs(matrix\_copy[i][k])**;** coord\_str = i**;** }  
 }  
  
 //Свап строк матрицы  
 if(max > std::abs(matrix\_copy[k][k])) {  
 std::swap(matrix\_copy[k]**,** matrix\_copy[coord\_str])**;** std::swap(b\_copy[k]**,** b\_copy[coord\_str])**;** }  
  
  
 for(int i = k + 1**;** i < b.size()**;** ++i) {  
 double l = matrix\_copy[i][k] / matrix\_copy[k][k]**;** matrix\_copy[i][k] = 0.0f**;** for(int j = k + 1**;** j < b.size()**;** ++j) {  
 matrix\_copy[i][j] -= l \* matrix\_copy[k][j]**;** }  
 b\_copy[i] -= l \* b\_copy[k]**;** }  
 }  
  
 //Обратный ход  
 std::vector<double> x(b.size()**,** 0)**;** x[b.size() - 1] = b\_copy[b.size() - 1] / matrix\_copy[b.size() - 1][b.size() - 1]**;** for(int i = b.size() - 2**;** i >= 0**;** --i) {  
 double sum = 0.0f**;** for(int j = i + 1**;** j < b.size()**;** ++j) {  
 sum += matrix\_copy[i][j] \* x[j]**;** }  
 x[i] = (b\_copy[i] - sum) / matrix\_copy[i][i]**;** }  
  
 return x**;**}  
  
void print5(std::vector<double> vector) {  
 for (int i = 0**;** i < 5**;** i++)  
 std::cout << vector[i] << " "**;** std::cout << "\n"**;**}  
  
double vectorNevyazki(std::vector<std::vector<double>> matrix**,** std::vector<double> x\_**,** std::vector<double> b) {  
 std::vector<double> b\_ = multiply(matrix**,** x\_)**;** double max = -1**;** for (int i = 0**;** i < b.size()**;** i++) {  
 max = std::max(std::abs(b\_[i] - b[i])**,** max)**;** }  
 return max**;**}  
  
double relativeError(std::vector<double> x\_**,** std::vector<double> x) {  
 double max\_1 = -1**;** for (int i = 0**;** i < x.size()**;** i++) {  
 max\_1 = std::max(std::abs(x\_[i] - x[i])**,** max\_1)**;** }  
 double max\_2 = -1**;** for (int i = 0**;** i < x.size()**;** i++) {  
 max\_2 = std::max(std::abs(x[i])**,** max\_2)**;** }  
 return max\_1 / max\_2**;**}  
  
  
int main()  
{  
 int size = 1700**;** std::vector<std::vector<double>> matrix = generateMatrix(size)**;** std::vector<double> x = generateVectorB(size)**;** std::vector<double> b = multiply(matrix**,** x)**;** auto start = std::chrono::steady\_clock::now()**;** std::vector<double> x\_ = gaussWithoutMainElem(matrix**,** b)**;** auto end = std::chrono::steady\_clock::now()**;** const int time1 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()**;** start = std::chrono::steady\_clock::now()**;** std::vector<double> y = gaussWithMainElem(matrix**,** b)**;** end = std::chrono::steady\_clock::now()**;** const int time2 = std::chrono::duration\_cast<std::chrono::milliseconds>(end - start).count()**;** std::cout << "Without choosing element" << "\n"**;** std::cout << "5 cordinats of solution "**;** print5(x\_)**;** std::cout << "Norma vectora nevyazki " << vectorNevyazki(matrix**,** x\_**,** b) << "\n"**;** std::cout << "otnositelnaya pogreshnost " << relativeError(x\_**,** x) << "\n"**;** std::cout << "Runtime " << time1 << " ms\n"**;** std::cout << "gauss with choosing main element\n"**;** std::cout << "5 cordinats of solution "**;** print5(y)**;** std::cout << "Norma vectora nevyazki " << vectorNevyazki(matrix**,** y**,** b) << "\n"**;** std::cout << "Otnositelnaya pogreshnost " << relativeError(y**,** x) << "\n"**;** std::cout << "Runtime " << time2 << " ms\n"**;** return 0**;**}

**Выходные данные.**

Without choosing element

5 cordinats of solution 14 15 16 17 18

Norma vectora nevyazki 0.107256

otnositelnaya pogreshnost 4.61083e-09

Runtime 10428 ms

gauss with choosing main element

5 cordinats of solution 14 15 16 17 18

Norma vectora nevyazki 7.45058e-06

Otnositelnaya pogreshnost 1.46778e-12

Runtime 10607 ms

Для типа данных Float:

5 cordinats of solution 836.252 -521.309 -150.051 92.2115 -385.229

Norma vectora nevyazki 1.3425e+07

otnositelnaya pogreshnost 0.847356

Runtime 10557 ms

gauss with choosing main element

5 cordinats of solution 13.504 15.0037 16.1465 16.7832 18.0113

Norma vectora nevyazki 3938.25

Otnositelnaya pogreshnost 0.000531466

Runtime 10841 ms

**Выводы.**

Метод Гаусса является неустойчивым: в случае относительной малости главного элемента результат вычисления получается с большой погрешностью. В то же время метод Гаусса с выбором ведущего элемента является устойчивым. Нельзя не отметить тот факт, что при вычислениях присутствует компьютерная погрешность, которая также приводит к неточностям. Выбор главного элемента не сильно влияет на время вычислений. При использовании типа данных float погрешность сильно увеличивается, что приводит к совершенно другому ответу. Это происходит из-за специфики хранения чисел типа float в памяти компьютера.