Министерство образования и науки Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

**Отчет по лабораторной работе**

**Решение систем линейных алгебраических уравнений**

**Выполнил**:студент группы 381606-2

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Тимакин Н.Е.

Подпись

**Проверил**: к.ф.-м.н., доц.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Баркалов К.А.

Подпись

Нижний Новгород 2017

**Введение**

В различных расчётах, в частности в линейной алгебре, может понадобиться решение систем алгебраических уравнений. Одним из алгоритмов их решения является метод Гауса.

**Постановка задачи**

Реализовать метод Гауса для решения систем линейных алгебраических уравнений в двух видах: первый с помощью типа данных double, а второй с помощью самостоятельно написанного класса рациональных чисел (rational). Сравнить результаты работы этих двух программ.

**Описание структуры программы**

Обе программы в целом идентичны за исключением того, что в варианте с рациональными числами описан класс rational, перегружены некоторые функции для этого класса, и все переменные типа double заменены на объекты класса rational.

Вариант double:

Программа состоит из 4 модулей:

funcs.h – заголовочный файл, здесь объявлены прототипы функций:

double \*createarr(int n); - создание массива

void arrfilling(double \*mas, int n); - заполнение массива

void printarray(double \*mas, int n); - печать массива

double \*\*create2darr(int n); - создание двумерного массива

void ddarrfilling(double \*\*mas, int n); - заполнение двумерного массива

void print2darr(double \*\*A, int n); - печать двумерного массива

double scal(double \*u, double \*v, int n); - скалярное произведение

double \*MnV(double \*\*A, double \*v, double n); - произведение матрицы на вектор

double \*\*transpl(double \*\*A, int n); - трансплонирование матрицы

void matrstrswap(double \*\*A, int n, int i, int j); - перемена мест i-ой и j-ой строк

int maxel(double \*\*A, int n, int j); - поиск максимального элемента в столбце матрицы

void GausMethod(double \*\*A, double \*b, double \*x, int n); - метод Гауса

void printsytem(double \*\*A, double \*b, int n); - печать СЛАУ

double \*\*copy2darr(double \*\*A, int n); - копирование двумерного массива

out.cpp – реализация функций ввода/вывода

vectors.cpp – реализация функций, связанных с алгебраическими операциями, в том числе и сам метод Гауса

main.cpp – главный модуль, содержащий функцию main

Вариант rational:

Программа состоит из 7 модулей:

funcs.h – заголовочный файл, здесь объявлены прототипы функций:

rational \*createarr(int n); - создание массива

void arrfilling(rational \*mas, int n); - заполнение массива

void printarray(rational \*mas, int n); - печать массива rational \*\*create2darr(int n); - создание двумерного массива

void ddarrfilling(rational \*\*mas, int n); - заполнение двумерного массива

void print2darr(rational \*\*A, int n); - печать двумерного массива

rational scal(rational \*u, rational \*v, int n); - скалярное произведение

rational \*MnV(rational \*\*A, rational \*v, int n); - произведение матрицы на вектор

rational \*\*transpl(rational \*\*A, int n); трансплонирование матрицы

void matrstrswap(rational \*\*A, int n, int i, int j); - перемена мест i-ой и j-ой строк

int maxel(rational \*\*A, int n, int j); - поиск максимального элемента в столбце матрицы

void GausMethod(rational \*\*A, rational \*b, rational \*x, int n); - метод Гауса

void printsytem(rational \*\*A, rational \*b, int n); - печать СЛАУ

rational \*\*copy2darr(rational \*\*A, int n); - копирование двумерного массива

out.cpp – реализация функций ввода/вывода

vectors.cpp – реализация функций, связанных с алгебраическими операциями, в том числе и сам метод Гауса

main.cpp – главный модуль, содержащий функцию main

rational.h – содержит объявления полей и методов класса rational

Скрытые поля m, n – числитель и знаменатель

Скрытый метод NOD() – вычисляет наибольший общий делитель дроби

rational(); - конструктор по умолчанию

rational(int a, int b); - конструктор-инициализатор

rational(int a); - конструктор преобразования типа

int getm(); - возвращает числитель

void setm(int par); - изменение числителя

int getn(); - возвращает знаменатель

void setn(int par) ; - изменение знаменателя

void printr(); - печать рационального числа

friend ostream& operator<<(ostream& os, const rational& p) ; - печать рационального числа с помощью потока вывода

friend istream& operator>>(istream& is, rational& p); - ввод рационального числа с помощью потока

void operator+=(rational b);

rational operator+(rational b);

void operator-=(rational b);

rational operator-(rational b);

void operator\*=(rational b);

rational operator\*(rational b);

void operator/=(rational b);

rational operator/(rational b);

friend rational operator+(int a, rational q);

friend rational operator-(int a, rational q);

friend rational operator\*(int a, rational q);

friend rational operator/(int a, rational q);

void operator+=(int b);

rational operator+(int b);

void operator-=(int b);

rational operator-(int b);

void operator\*=(int b);

rational operator\*(int b);

void operator/=(int b);

rational operator/(int b); - различные арифметические операции с рациональными числами

void random(); - задаёт случайные значения числителю и знаменателю

bool operator<(const rational q) const;

bool operator<=(const rational q) const;

bool operator>(const rational q) const;

bool operator>=(const rational q) const;

bool operator==(const rational q) const;

bool operator!=(const rational q) const; - различные логические операции с рациональными числами

friend rational fabs(rational p); - модуль рационального числа

friend double sqrt(rational p); - квадратный корень из рационального числа

rational.cpp – модуль, содержащий реализацию методов класса rational

R\_operations.cpp – модуль, содержащий реализацию некоторых функций класса rational, не являющихся методами класса

**Описание алгоритмов**

Ввиду идентичности алгоритмов будут приведены описания только для варианта rational

rational \*createarr(int n); - создаётся указатель на rational и под него выделяется память

void arrfilling(rational \*mas, int n); - если n<=3, то в цикле ввести в поток значения mas, иначе задать случайные значения

void printarray(rational \*mas, int n); - через цикл вывести в поток каждый элемент mas

rational \*\*create2darr(int n); - создаётся двойной указатель на rational, под него выделяется память. Через цикл память выделяется под каждый одинарный указатель

void ddarrfilling(rational \*\*mas, int n); - в цикле для каждого элемента mas вызывается функция arrfilling

void print2darr(rational \*\*A, int n); - в цикле для каждого элемента А вызывается функция printarray

rational scal(rational \*u, rational \*v, int n); - создаётся результат, приравнивается к 0, в него записывается через цикл сумма произведений координат векторов u и v

rational \*MnV(rational \*\*A, rational \*v, int n); - создаётся указатель на rational и для него вызывается функция createarr. Через цикл для каждому элементу этого массива присваивается значение скалярного произведения i-ой строки матрицы и вектора v

rational \*\*transpl(rational \*\*A, int n); - создаётся двойной указатель на rational, ему присваивается значение функции create2darr, через двойной цикл по i и j В[i][j] присваивается A[j][i]

void matrstrswap(rational \*\*A, int n, int i, int j); - создаётся указатель на rational, ему присваивается первая строка, первой строке присваивается вторая, второй - временная

int maxel(rational \*\*A, int n, int j); - создаётся переменная int i=j и max типа rational, ей присваивается значение модуля A[j][j]. Через цикл по k, если A[k][j]>max, то max присваивается модуль A[k][j], а i присваивается k

void GausMethod(rational \*\*A, rational \*b, rational \*x, int n); - создаётся рациональный коэффициент. Если текущий элемент на диагонали равен 0, то текущая строка меняется со строкой с максимальным элементом в текущем столбце. Если текущий элемент на диагонали не равен 0, то из всех последующих строк вычитается предыдущая, умноженная на коэффициент, и при каждом действии печатается текущий вид системы. Выполняется проверка, вычислением погрешности между векторами

void printsytem(rational \*\*A, rational \*b, int n); - если размер больше 10, то не печатать систему, иначе в цикле вызвать функцию printarray для каждой строки и вывести в поток каждый элемент вектора b

rational \*\*copy2darr(rational \*\*A, int n); - создаётся двойной указатель на rational, ему присваивается значение функции create2darr, через двойной цикл каждому элементу первого двумерного массива присваивается элемент второго

Далее будут представлены алгоритмы арифметических операций, которые будут приведены только для одной операции для каждого вида

void operator+=(rational b); - первому числителю присваивается сумма произведений первого числителя на второй знаменатель и второго числителя на первый знаменатель. Первому знаменателю присваивается произведение двух знаменателей. Применяется метод НОД

rational operator-(rational b); - создаётся рациональный результат. Числителю результата присваивается разность произведений первого числителя на второй знаменатель и второго числителя на первый знаменатель. Знаменателю результата присваивается произведение двух знаменателей. Применяется метод НОД. Возвращается результат

friend rational operator+(int a, rational q); - возвращается разность a и q

void operator\*=(int b); - первый числитель домножается на второй числитель, и первый знаменатель домножется на второй знаменатель. Применяется метод НОД

rational operator/(int b); - создаётся рациональный результат. Числителю результата присваивается произведение первого числителя на второй знаменатель. Знаменателю результата присваивается произведение первого знаменателя на второй числитель. Применяется метод НОД. Возвращается результат

Логические операции на примере операции меньше:

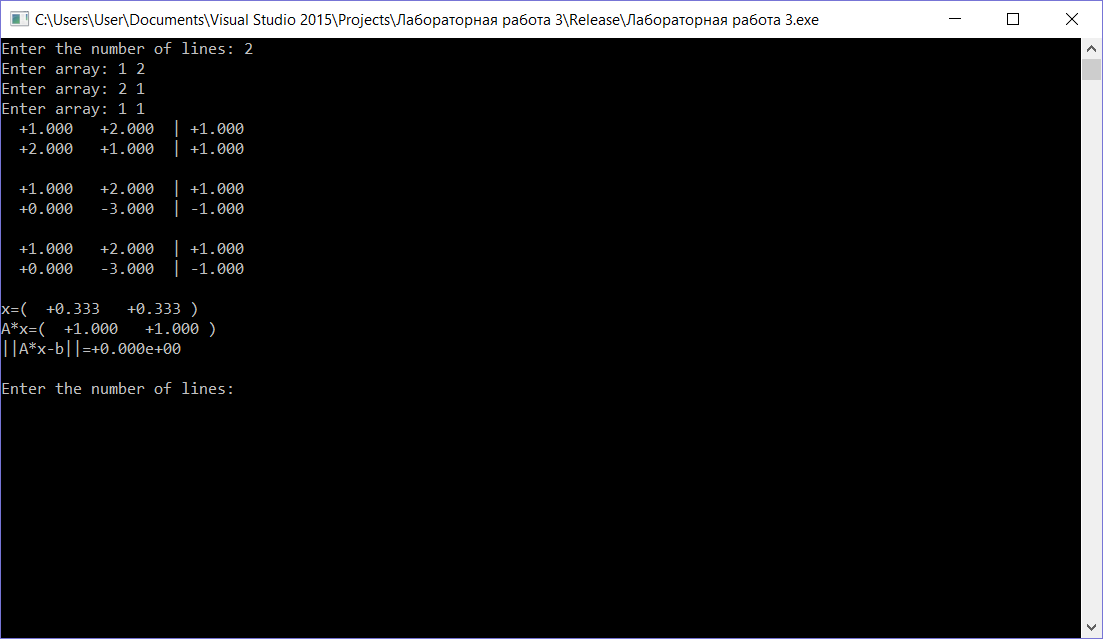
bool operator<(const rational q) const; - создаётся рациональный результат. Числителю результата присваивается разность произведений числителя q на первый знаменатель и знаменателя q на первый числитель. Знаменателю результата присваивается произведение знаменателей. Если числитель и знаменатель результата оба больше 0 или оба меньше нуля, то вернуть истину, иначе вернуть ложь

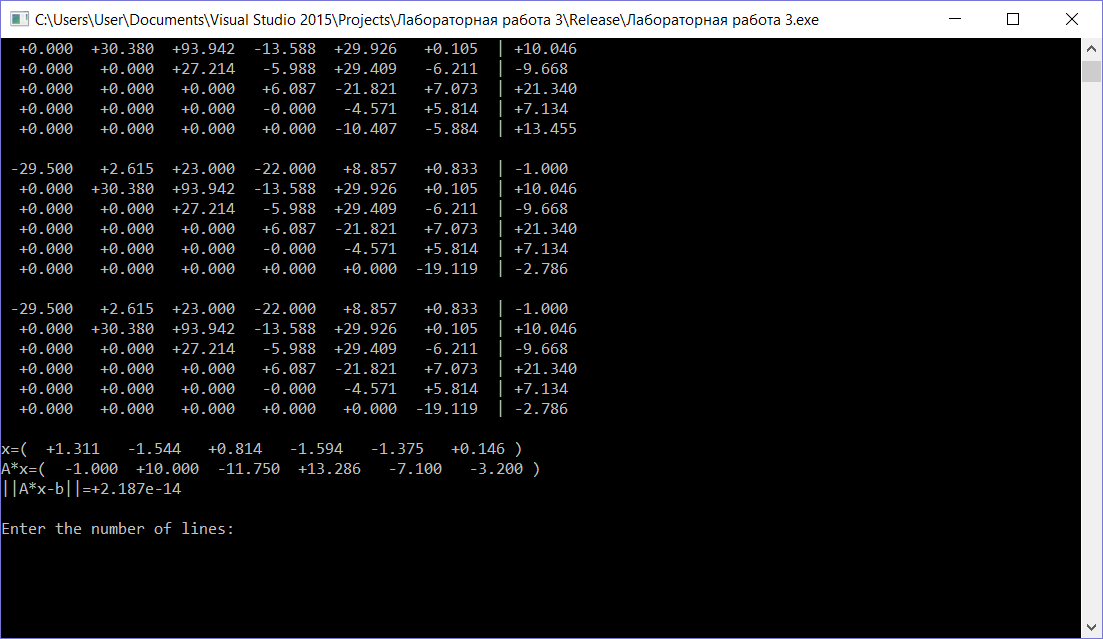
friend rational fabs(rational p); - создаётся рациональный результат. Числителю результата присваивается модуль числителя q. Знаменателю результата присваивается модуль знаменателя q

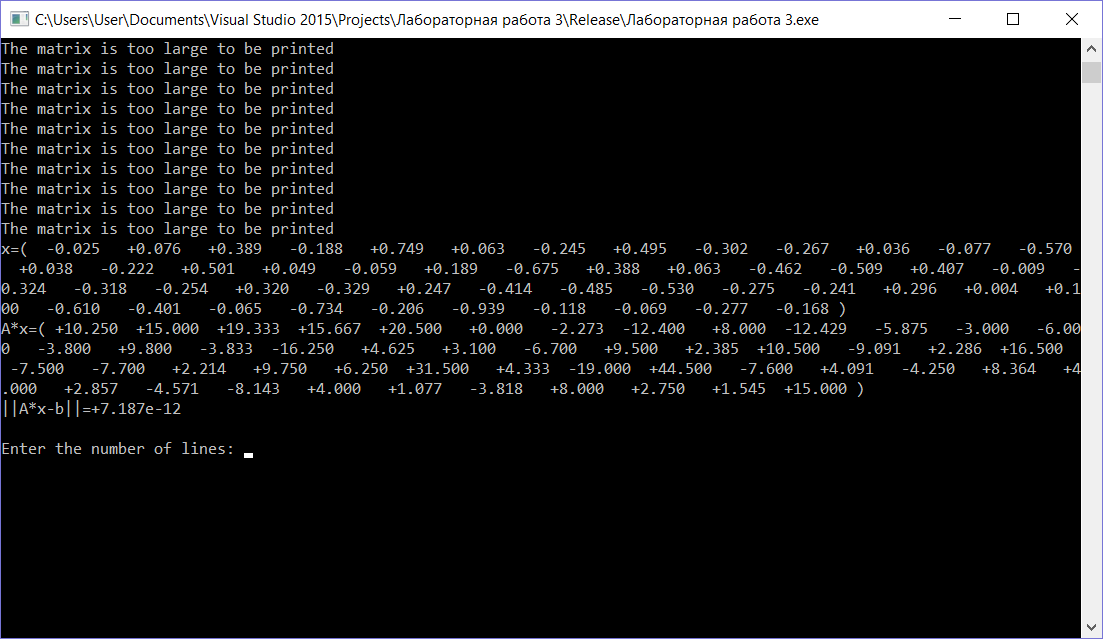
friend double sqrt(rational p); - результату присваивается частное корней из числителя и из знаменателя

**Результаты**

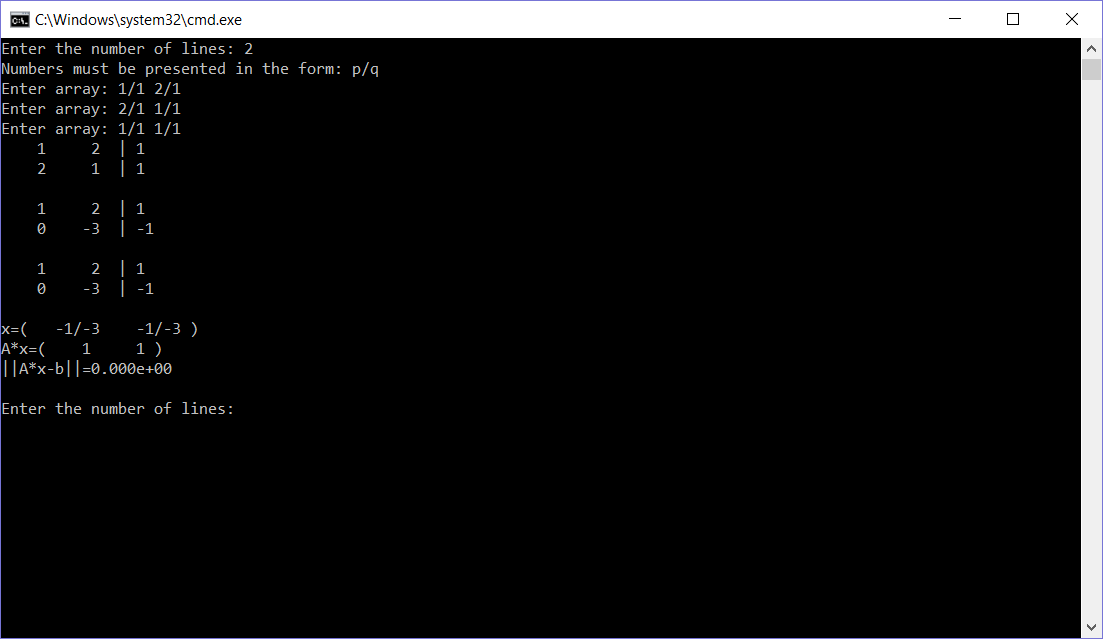
**Версия double:**

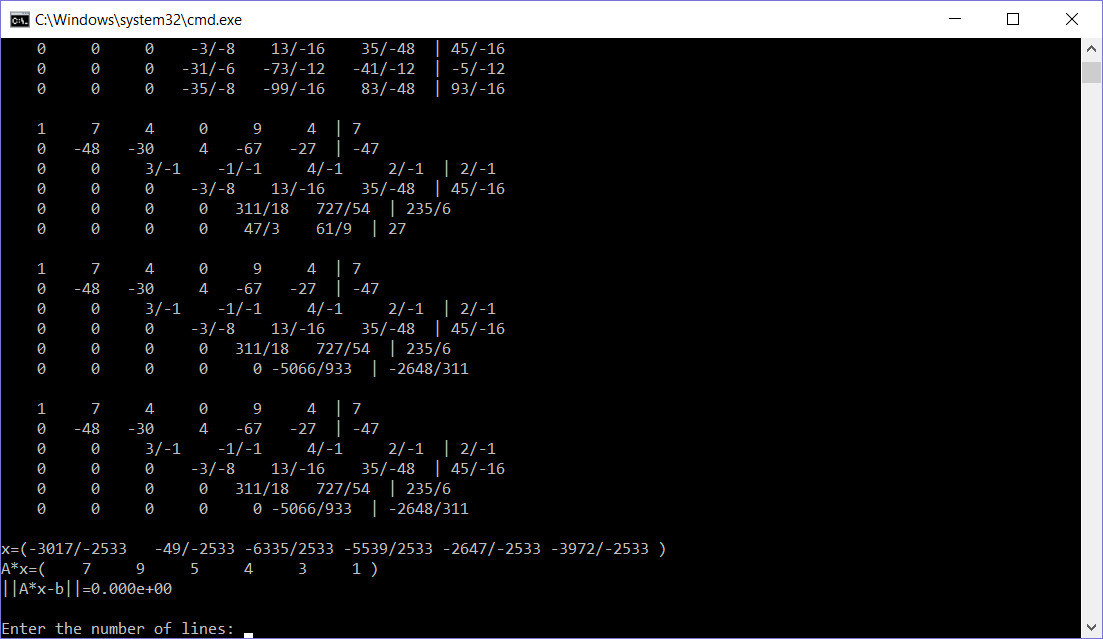






**Версия rational:**







**Вывод**

Как видно из результатов работы обеих программ, при малом количестве переменных, например при двух, обе выдают одинаково точные результаты без погрешности. При увеличении количества переменных в варианте double увеличивается погрешность, в отличие от варианта rational, там выдаётся точный результат. При дальнейшем увеличении количества переменных погрешность в варианте double возрастает, а в варианте rational системы уравнений, где больше 7 переменных, уже не решаются, так как в них наступает переполнение типа int. Так что решение о том, какую программу использовать для вычислений, нужно принимать, исходя из того, сколько в данной системе переменных

**Литература**

Брайан Керниган, Деннис Ритчи «Язык программирования Си»

Брюс Эккель «Философия С++. Введение в стандартный С++»

Стивен Прата «Язык программирования С++. Лекции и упражнения»

**Приложение**

Вариант double:

funcs.h

#pragma once

#include <iostream>

using namespace std;

double \*createarr(int n);

void arrfilling(double \*mas, int n);

void printarray(double \*mas, int n);

double \*\*create2darr(int n);

void ddarrfilling(double \*\*mas, int n);

void print2darr(double \*\*A, int n);

double scal(double \*u, double \*v, int n);

double \*MnV(double \*\*A, double \*v, double n);

double \*\*transpl(double \*\*A, int n);

void matrstrswap(double \*\*A, int n, int i, int j);

int maxel(double \*\*A, int n, int j);

void GausMethod(double \*\*A, double \*b, double \*x, int n);

void printsytem(double \*\*A, double \*b, int n);

double \*\*copy2darr(double \*\*A, int n);

out.cpp

#include "funcs.h"

double \*createarr(int n)

{

double \*mas = new double[n];

return mas;

}

void arrfilling(double \*mas, int n)

{

if (n <= 5)

{

cout << "Enter array: ";

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

{

cin >> mas[i];

}

}

else

{

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

mas[i] = (double)(rand() % 200-100)/(rand()%14+1);

}

}

void printarray(double \*mas, int n)

{

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

{

cout.setf(ios::fixed);

//cout.setf(ios::scientific);

cout.setf(ios::showpos);

cout.setf(ios::right);

cout.width(8);

cout.precision(3);

//cout.fill('&');

cout << mas[i] << " ";

}

}

double \*\*create2darr(int n)

{

double \*\*A = new double \*[n];

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

{

A[i] = new double[n];

}

return A;

}

void ddarrfilling(double \*\*A, int n)

{

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

{

arrfilling(A[i], n);

}

}

void print2darr(double \*\*A, int n)

{

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

{

printarray(A[i], n);

cout << endl;

}

cout << endl;

}

void printsytem(double \*\*A, double \*b, int n)

{

if (n > 10)

{

cout << "The matrix is too large to be printed\n";

return ;

}

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

{

printarray(A[i], n);

cout << " | " << b[i] << endl;

}

cout << endl;

}

vectors.cpp

#include "funcs.h"

double scal(double \*u, double \*v, int n)

{

double res = 0;

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

{

res += v[i] \* u[i];

}

return res;

}

double \*MnV(double \*\*A, double \*v, double n)

{

double \*a = createarr(n);

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

{

a[i] = scal(A[i], v, n);

}

return a;

}

double \*\*transpl(double \*\*A, int n)

{

double \*\*B = create2darr(n);

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

{

for (int j = 0; j < n; j++)

{

B[i][j] = A[j][i];

}

}

return B;

}

void matrstrswap(double \*\*A, int n, int i, int j)

{

double \*temp;

temp = A[i];

A[i] = A[j];

A[j] = temp;

}

int maxel(double \*\*A, int n, int j)

{

double max =fabs( A[j][j]);

int i = j;

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

{

if (fabs(A[k][j]) > max)

{

max = fabs(A[k][j]);

i = k;

}

}

return i;

}

void GausMethod(double \*\*A, double \*b, double \*x, int n)

{

int i, j, k;

double alfa;

for (j = 0; j < n; j++)

{

if (!A[j][j])

{

int nmax=maxel(A, n, j);

//cout << "nmax=" << nmax << endl;

matrstrswap(A, n, j, nmax);

swap(b[j], b[nmax]);

}

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

{

alfa = A[i][j] / A[j][j];

for (k = j; k < n; k++)

{

A[i][k] -= alfa\*A[j][k];

}

b[i] -= alfa\*b[j];

}

printsytem(A, b, n);

}

double sum = 0;

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

{

sum = 0;

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

{

sum += A[i][j] \* x[j];

}

x[i] = (b[i] - sum) / A[i][i];

}

}

double \*\*copy2darr(double \*\*A, int n)

{

double \*\*B = create2darr(n);

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

for (int j = 0; j < n; j++)

B[i][j] = A[i][j];

return B;

}

main.cpp

#include "funcs.h"

int main()

{

while(1)

{

cout << "Enter the number of lines: ";

int n;

cin >> n;

if (!n) break;

double \*\*A1 = create2darr(n);

ddarrfilling(A1, n);

double \*\*A = copy2darr(A1, n);

double \*b1 = new double[n];

arrfilling(b1, n);

double \*b = new double[n];

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

b[i] = b1[i];

double \*x = new double[n];

printsytem(A, b, n);

GausMethod(A, b, x, n);

cout << "x=(";

printarray(x, n);

cout << ")\n";

cout << "A\*x=(";

double \*y = new double[n];

y = MnV(A1, x, n);

printarray(y, n);

cout << ")\n";

double \*z = new double[n];

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

{

z[i] = y[i] - b1[i];

}

cout << "||A\*x-b||=" << scientific << sqrt(scal(z, z, n)) << fixed << endl << endl;

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

{

delete[] A[i];

}

delete[] A;

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

{

delete[] A1[i];

}

delete[] A1;

delete[] b;

delete[] b1;

delete[] x;

delete[] y;

delete[] z;

}

return 0;

}

Вариант rational:

funcs.h

#pragma once

#include "rational.h"

rational \*createarr(int n);

void arrfilling(rational \*mas, int n);

void printarray(rational \*mas, int n);

rational \*\*create2darr(int n);

void ddarrfilling(rational \*\*mas, int n);

void print2darr(rational \*\*A, int n);

rational scal(rational \*u, rational \*v, int n);

rational \*MnV(rational \*\*A, rational \*v, int n);

rational \*\*transpl(rational \*\*A, int n);

void matrstrswap(rational \*\*A, int n, int i, int j);

int maxel(rational \*\*A, int n, int j);

void GausMethod(rational \*\*A, rational \*b, rational \*x, int n);

void printsytem(rational \*\*A, rational \*b, int n);

rational \*\*copy2darr(rational \*\*A, int n);

out.cpp

#include "funcs.h"

rational \*createarr(int n)

{

rational \*mas = new rational[n];

return mas;

}

void arrfilling(rational \*mas, int n)

{

if (n <= 3)

{

cout << "Enter array: ";

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

{

cin >> mas[i];

}

}

else

{

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

mas[i].random();

}

}

void printarray(rational \*mas, int n)

{

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

{

cout.setf(ios::fixed);

//cout.setf(ios::scientific);

//cout.setf(ios::showpos);

cout.setf(ios::right);

cout.width(5);

cout.precision(3);

//cout.fill('&');

cout << mas[i] << " ";

}

}

rational \*\*create2darr(int n)

{

rational \*\*A = new rational \*[n];

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

{

A[i] = new rational[n];

}

return A;

}

void ddarrfilling(rational \*\*A, int n)

{

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

{

arrfilling(A[i], n);

}

}

void print2darr(rational \*\*A, int n)

{

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

{

printarray(A[i], n);

cout << endl;

}

cout << endl;

}

void printsytem(rational \*\*A, rational \*b, int n)

{

if (n > 10)

{

cout << "The matrix is too large to be printed\n";

return ;

}

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

{

printarray(A[i], n);

cout << " | " << b[i] << endl;

}

cout << endl;

}

vectors.cpp

#include "funcs.h"

rational scal(rational \*u, rational \*v, int n)

{

rational res = 0;

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

{

res += v[i] \* u[i];

}

return res;

}

rational \*MnV(rational \*\*A, rational \*v, int n)

{

rational \*a = createarr(n);

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

{

a[i] = scal(A[i], v, n);

}

return a;

}

rational \*\*transpl(rational \*\*A, int n)

{

rational \*\*B = create2darr(n);

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

{

for (int j = 0; j < n; j++)

{

B[i][j] = A[j][i];

}

}

return B;

}

void matrstrswap(rational \*\*A, int n, int i, int j)

{

rational \*temp;

temp = A[i];

A[i] = A[j];

A[j] = temp;

}

int maxel(rational \*\*A, int n, int j)

{

rational max = fabs(A[j][j]);

int i = j;

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

{

if (fabs(A[k][j]) > max)

{

max = fabs(A[k][j]);

i = k;

}

}

return i;

}

void GausMethod(rational \*\*A, rational \*b, rational \*x, int n)

{

int i, j, k;

rational alfa;

for (j = 0; j < n; j++)

{

if (A[j][j]==0)

{

int nmax=maxel(A, n, j);

//cout << "nmax=" << nmax << endl;

matrstrswap(A, n, j, nmax);

swap(b[j], b[nmax]);

}

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

{

alfa = A[i][j] / A[j][j];

for (k = j; k < n; k++)

{

A[i][k] -= alfa\*A[j][k];

}

b[i] -= alfa\*b[j];

}

printsytem(A, b, n);

}

rational sum = 0;

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

{

sum = 0;

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

{

sum += A[i][j] \* x[j];

}

x[i] = (b[i] - sum) / A[i][i];

}

}

rational \*\*copy2darr(rational \*\*A, int n)

{

rational \*\*B = create2darr(n);

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

for (int j = 0; j < n; j++)

B[i][j] = A[i][j];

return B;

}

main.cpp

#include "funcs.h"

int main() {

while (1)

{

cout << "Enter the number of lines: ";

int n;

cin >> n;

if (!n) break;

cout << "Numbers must be presented in the form: p/q " << endl;

rational \*\*A1 = create2darr(n);

ddarrfilling(A1, n);

rational \*\*A = copy2darr(A1, n);

rational \*b1 = new rational[n];

arrfilling(b1, n);

rational \*b = new rational[n];

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

b[i] = b1[i];

rational \*x = new rational[n];

printsytem(A, b, n);

GausMethod(A, b, x, n);

cout << "x=(";

printarray(x, n);

cout << ")\n";

cout << "A\*x=(";

rational \*y = new rational[n];

y = MnV(A1, x, n);

printarray(y, n);

cout << ")\n";

rational \*z = new rational[n];

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

{

z[i] = y[i] - b1[i];

}

cout << "||A\*x-b||=" << scientific << sqrt(scal(z, z, n)) << fixed << endl << endl;

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

{

delete[] A[i];

}

delete[] A;

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

{

delete[] A1[i];

}

delete[] A1;

delete[] b;

delete[] b1;

delete[] x;

delete[] y;

delete[] z;

}

return 0;

}

rational.h

#pragma once

#include <iostream>

using namespace std;

class rational {

private:

int m, n;

void NOD();

public:

rational() {

m = 0;

n = 1;

}

rational(int a, int b) {

m = a;

n = b;

NOD();

}

rational(int a) {

m = a;

n = 1;

}

int getm() {

return m;

}

void setm(int par) {

m = par;

}

int getn() {

return n;

}

void setn(int par) {

n = par;

}

void printr();

friend ostream& operator<<(ostream& os, const rational& p) {

if (p.m == 0 || p.n == 1)

os << p.m;

else

os << p.m << "/" << p.n;

return os;

}

friend istream& operator>>(istream& is, rational& p) {

char slash;

is >> p.m >> slash >> p.n;

return is;

}

void operator+=(rational b);

rational operator+(rational b);

void operator-=(rational b);

rational operator-(rational b);

void operator\*=(rational b);

rational operator\*(rational b);

void operator/=(rational b);

rational operator/(rational b);

void operator+=(int b);

rational operator+(int b);

void operator-=(int b);

rational operator-(int b);

void operator\*=(int b);

rational operator\*(int b);

void operator/=(int b);

rational operator/(int b);

void random();

bool operator<(const rational q) const;

bool operator<=(const rational q) const;

bool operator>(const rational q) const;

bool operator>=(const rational q) const;

bool operator==(const rational q) const;

bool operator!=(const rational q) const;

friend rational operator+(int a, rational q);

friend rational operator-(int a, rational q);

friend rational operator\*(int a, rational q);

friend rational operator/(int a, rational q);

friend rational fabs(rational p);

friend double sqrt(rational p);

};

rational.cpp

#include "rational.h"

void rational::printr() {

if((m == 0) || (n == 1))

cout << m;

else

cout << m << "/" << n;

}

void rational::operator+=(rational b) {

m = m\*b.n + n\*b.m;

n \*= b.n;

NOD();

}

rational rational::operator+(rational b) {

rational res;

res.m = m\*b.n + n\*b.m;

res.n = n \* b.n;

res.NOD();

return res;

}

void rational::operator-=(rational b) {

m = m\*b.n - n\*b.m;

n \*= b.n;

NOD();

}

rational rational::operator-(rational b) {

rational res;

res.m = m\*b.n - n\*b.m;

res.n = n \* b.n;

res.NOD();

return res;

}

void rational::operator\*=(rational b) {

m \*= b.m;

n \*= b.n;

NOD();

}

rational rational::operator\*(rational b) {

rational res;

res.m = m \* b.m;

res.n = n \* b.n;

res.NOD();

return res;

}

void rational::operator/=(rational b) {

m \*= b.n;

n \*= b.m;

NOD();

}

rational rational::operator/(rational b) {

rational res;

res.m = m \* b.n;

res.n = n \* b.m;

res.NOD();

return res;

}

void rational::NOD() {

int \_m = abs(m), \_n = abs(n);

while ((\_m != 0) && (\_n != 0)) {

if (\_m > \_n)

\_m %= \_n;

else

\_n %= \_m;

}

int nod = \_m + \_n;

m /= nod;

n /= nod;

}

void rational::operator+=(int b) {

m += b\*n;

NOD();

}

rational rational::operator+(int b) {

rational res;

res.m = m + b\*n;

res.n = n;

NOD();

return res;

}

void rational::operator-=(int b) {

m -= b\*n;

NOD();

}

rational rational::operator-(int b) {

rational res;

res.m = m - b\*n;

res.n = n;

NOD();

return res;

}

void rational::operator\*=(int b) {

m \*= b;

NOD();

}

rational rational::operator\*(int b) {

rational res;

res.m = m\*b;

res.n = n;

NOD();

return res;

}

void rational::operator/=(int b) {

n \*= b;

NOD();

}

rational rational::operator/(int b) {

rational res;

res.m = m;

res.n = n\*b;

NOD();

return res;

}

void rational::random() {

m = rand() % 10;

n = 1;

}

bool rational::operator<(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m > 0 && res.n > 0) || (res.m < 0 && res.n < 0))

return true;

else

return false;

}

bool rational::operator<=(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m >= 0 && res.n > 0) || (res.m <= 0 && res.n < 0))

return true;

else

return false;

}

bool rational::operator>(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m < 0 && res.n > 0) || (res.m > 0 && res.n < 0))

return true;

else

return false;

}

bool rational::operator>=(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m <= 0 && res.n > 0) || (res.m >= 0 && res.n < 0))

return true;

else

return false;

}

bool rational::operator==(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m == 0 && res.n > 0) || (res.m == 0 && res.n < 0))

return true;

else

return false;

}

bool rational::operator!=(const rational q) const {

rational res;

res.m = q.m\*n - m\*q.n;

res.n = q.n\*n;

if ((res.m != 0 && res.n > 0) || (res.m != 0 && res.n < 0))

return true;

else

return false;

}

R\_operations.cpp

#include "funcs.h"

rational operator+(int a, rational q) {

return q + a;

}

rational operator-(int a, rational q) {

return -1 \* (q - a);

}

rational operator\*(int a, rational q) {

return q \* a;

}

rational operator/(int a, rational q) {

rational res;

res.m = a\*q.n;

res.n = q.m;

res.NOD();

return res;

}

rational fabs(rational p) {

rational res;

res.m = abs(p.m);

res.n = abs(p.n);

return res;

}

double sqrt(rational p) {

double res = sqrt(p.m) / sqrt(p.n);

return res;

}